Thabet Slimani

ALGORITHMIQUE ET STRUCTURES DE DONNEES

Cours et exercices corrigés

EDITIONS CLE

Thabet Slimani

Algorithmique et structures de données I : Note de cours, exercices corrigés et examens types

Editions C.L.E

I.S.B.N. – 978-9973 -45-069-2
Algorithmique et Structures des Données I :
Notes de Cours, exercices Corrigés et examens Types.
© Editions C.L.E. Contribution à la Littérature d'Entreprise 2011
96. Rue de Yougoslavie 1001 Tunis
Tél. : 71.257.680- Fax : 71.349-796
E.mail : editions.cle@gnet.tn
Prix : 15.000

Avant-propos

Connaissez-vous les mécanismes utilisés pour que votre ordinateur fasse des calculs pareils à vous ? Connaissez-vous l'exactitude du résultat affiché ? Combien de temps devrez-vous attendre la fin du calcul ? N'y a-t-il pas un moyen de l'obtenir plus vite ? Que vous soyez ingénieur, mathématicien, statisticien et surtout informaticien, toutes ces questions vous vous les posez. Si vous êtes étudiant et même enseignant, elles surgiront très rapidement.

Étudier l'algorithmique, c'est apporter des réponses à vos questions. Cette science est le cœur de l'informatique. Pour tous ceux qui doivent ou devront faire travailler un ordinateur, il est essentiel de comprendre ses principes fondamentaux et de connaître ses éléments de base. Une formule ne se conduit pas comme une voiture à pédales. De même, un ordinateur ne s'utilise pas comme un boulier. L'algorithmique est une discipline qui désigne le permis de conduire de l'informaticien. Sans elle, il n'est pas concevable d'exploiter sans risque un ordinateur.

L'absence d'un ouvrage universitaire d'algorithmique simple à l'utilisation par des étudiants débutants et la difficulté de la présentation des algorithmes par la majorité de ces ouvrages m'a conduit à introduire d'une manière simplifiée la théorie de l'algorithmique et des structures de données. Cette introduction sort un peu de l'ordinaire, surtout au niveau de la syntaxe de quelques instructions. La syntaxe suivie est proche à la programmation C ou Java, ce choix revient au fait que la majorité

des programmes universitaires concernant les pratiques de programmation utilisent le langage C ou Java.

Cette introduction remarquable à l'algorithmique donne au lecteur d'une part les bases théoriques indispensables et lui fournit d'autre part les moyens de concevoir rigoureusement des programmes efficaces permettant de résoudre des problèmes variés issus de différentes applications.

Les notions des algorithmes présentées dans cet ouvrage prennent en compte le niveau des étudiants débutants, en définissant les 4 briques de bases constituant un algorithme (les variables, les instructions élémentaires, les tests et les boucles), ainsi que la décomposition des algorithmes en sous-algorithmes. En plus, une petite place est réservée aux structures de données (tableau et matrices). Un ensemble d'algorithmes classiques utilisés dans le processus de tri et de recherche d'informations à partir d'un tableau est aussi présenté.

Il est certain que la plupart des informaticiens débutants trouveront dans ce livre les notions de base et l'appui nécessaire pour la rédaction d'un algorithme de base, ainsi que certaines avancées récentes dans leur domaine. Cet ouvrage met donc en relief le rôle central joué par l'algorithmique dans la science informatique. En plus, cet ouvrage présente 7 chapitres théoriques accompagnés par plus de 90 exercices corrigés. Et pour certains exercices, il existe des exemples d'exécutions manuelles permettant d'éclairer, à l'étudiant, la procédure d'exécution pour un ordinateur, ainsi que la vérification de l'exactitude de son algorithme. De même, l'avantage le plus remarquable pour cet ouvrage est l'inclusion des épreuves d'examens types accompagnées par des solutions.

Cet ouvrage s'adresse aux étudiants de la première année (cycle de licence (informatique et de mathématique, cycle préparatoire des ingénieurs). Tous y trouveront une aide et un support de cours utile tout au long de leurs études.

La diversité des sujets abordés, l'efficacité des algorithmes présentés, et leur écriture dans un pseudo-code proche du langage *C*, qui les rendent très faciles à implémenter, font aussi de ce livre un recueil fort utile dans la vie professionnelle d'un informaticien ou d'un ingénieur.

Enfin, au-delà de ses besoins propres, nous souhaitons que le lecteur, qu'il soit enseignant, ingénieur, étudiant, ou simplement curieux, prenne comme nous plaisir et intérêt à la lecture de cet ouvrage.

Thabet Slimani
Mars 2011

TABLE DES MATIÈRES

Chapitre I ***14***

Introduction Générale ***14***

1.1) Introduction à l'algorithmique **15**

1.1.1) Définition d'un algorithme 15

1.1.2) Rédaction d'un algorithme à partir d'un problème 17

1.1.3) Efficacité d'un algorithme 18

1.1.4) Utilité d'un algorithme 19

1.1.5) Sous-algorithmes 20

1.2) Utilisation de données **21**

1.2.1) Types de décision 22

1.2.2) Gestion de données 22

1.2.3) Objectifs de gestion de données 23

1.3) Introduction aux structures de données **23**

1.4) Programmation et langages informatiques **25**

1.4.1) Langage de programmation 26

a) langages de bas niveau 26

b) langages de haut niveau (langages évolués) 26

1.4.2) Un programme 27

1.4.3) Passage de l'algorithme au programme 28

Chapitre 2 ..... ***15***

Notions de base ***15***

2.1) Formalisme d'un algorithme **31**

2.2) La notion d'objet **32**

2.2.1) Définition d'une variable 33

2.2.2) Définition d'une constante 33

2.3) Types de données **34**

2.3.1) Les types scalaires 35

a) Les entiers 35

b) Les réels (nombres flottants) 37
2.3.2) Les types élémentaires 39
a) Les booléens 39
b) Les caractères 40
c) Les chaînes de caractères 40
d) Les types énumérés 40
e) Le type structuré 41

2.4) Les expressions et les opérateurs 41
2.4.1) Les opérateurs 42
2.4.2) Priorité des opérateurs 45

5) Les instructions élémentaires 46
2.5.1) Instruction/expression d'affectation 47
2.5.2) Instructions d'entrée/sortie 48
2.5.3) Exemple d'entrée/sortie 49
2.5.4) Règles pour la rédaction des instructions d'E/S 49

Chapitre 3 64

les structures conditionnelles 64

3.1) Rappels sur la logique booléenne 65
3.1.1) Les conditions simples 66
3.1.2) Les conditions complexes 67

3.2) Les structures de choix 68
3.2.1) Structure de choix simple 68
3.2.2) Structure de choix alternative 70
3.2.3) Structure de choix imbriquée 71

3.3) Les structures d'aiguillage (de cas) 73

Chapitre 4 102

les structures répétitives 102

4.1) Les itérations 103
4.1.1) Les itérations indéterministes 104
a) La structure tant que 104
b) La structure Répéter...jusqu'à 108

4.1.2) Les itérations déterministes (pour) 110

4.2) Les boucles imbriquées 115

4.3) Choix de la structure itérative 116

Chapitre 5 162

les sous-algorithmes 162

5.1) Les sous-algorithmes 164

5.2) Les sous-algorithmes : types et règles d'utilisation 165

5.2.1) Règles de nommage des sous-algorithmes 165

5.2.2) Les fonctions 166

5.2.3) Les procédures 168

5.2.4) Structure d'un algorithme principal 169

5.2.5) Définition de la portée des variables 170

5.2.6) Les paramètres formels/effectifs 172

a) Le passage de paramètres en entrée 173

b) Les passage de paramètres en sortie 174

c) Le passages de paramètres en entrée/sortie 174

d) Exemples de passage par paramètre 176

e) Circulation des informations dans un algorithme 179

5.3) Les fonctions récursives 180

Chapitre 6 211

la structure de données tableau 211

6.1) Utilité des tableaux 213

6.2) Tableau statique à une dimension 215

6.3) Opérations élémentaires sur les tableaux 216

6.3.1) Utilisation d'un tableau par les indices 216

6.3.2) Initialisation d'un tableau 216

6.3.3) Affichage des éléments d'un tableau 217

6.3.4) Ajout d'un élément à la fin d'un tableau 218

6.3.5) Recherche d'un élément dans un tableau 219

6.3.6) Recherche de la position d'un élément dans un tableau 220

6.3.7) Suppression d'un élément dans un tableau 221
6.3.8) Insertion d'un élément dans un tableau 222
6.3.9) Inversion de l'ordre des éléments d'un tableau 224

6.4) Applications sur les ensembles **225**

6.5) Les tableaux bidimensionnels **231**

Chapitre 7 ..... ***255***

les méthodes de tri et de recherche dans un tableau ***255***

7.1) Le tri d'un tableau **257**
7.1.1) Le tri naïf 258
7.1.2) Le tri par insertion 259
7.1.3) Le tri par sélection 265
7.1.4) Le tri à bulle (bubble sort) 271
7.1.5) Le tri par fusion 276

7.2) Les méthodes de recherches dans un tableau **280**
7.2.1) Recherche séquentielle 281
7.2.2) Recherche dichotomique 284

Examens corrigés ***290***

TABLE DES FIGURES

Figure 1. Communication de l'information entre un utilisateur et un algorithme *16*
Figure 2. Les éventuels appels entre les sous-algorithmes d'un problème *20*
Figure 3. *Les données : les ingrédients essentielles pour prendre des actions basées sur la prise de décision informatisée* *21*
Figure 4. *De l'information aux données binaires* *25*
Figure 5. Exécution d'un programme *28*
Figure 6. Schéma de communication des informations entre l'ordinateur et ses périphériques *47*
Figure 7 Schéma d'évolution de la structure tant que *105*
Figure 8. Schéma d'évolution de l'instruction répéter...jusqu'à *109*
Figure 9. Diagramme d'évolution de l'instruction Pour *112*
Figure 10. *Echange de données entre un algorithme principal et ses sous-algorithmes* *179*
Figure 11. Les étapes d'un exemple de tri par fusion *277*

LISTE DES TABLEAUX

Tableau 1. Borne supérieure des nombres entiers.....36
Tableau 2. Déclaration variée des entiers.....37
Tableau 3. Formats des flottants IEEE.....38
Tableau 4. Déclaration des réels.....39
Tableau 5 Les opérateurs booléens.....44
Tableau 6 Les d'opérateurs énumérés.....44
Tableau 7 Les d'opérateurs en algorithme et son équivalence en C.....45
Tableau 8. Syntaxe d'utilisation des fonctions d'entrée/sortie en algorithme.....50
Tableau 9. Les opérateurs de comparaison.....67
Tableau 10. Test à la main de l'algorithme plusGrandEntier (cas n=10).....107
Tableau 11. Exécution à la main de l'algorithme pgcd euclidien.....110
Tableau 12. Exécution à la main de l'algorithme SomnPremierEntier.....114

Chapitre I

Introduction Générale

1.1) Introduction à l'algorithmique 15
1.1.1) Définition d'un algorithme 15
1.1.2) Rédaction d'un algorithme à partir d'un problème 17
1.1.3) Efficacité d'un algorithme 18
1.1.4) Utilité d'un algorithme 19
1.1.5) Sous-algorithmes 20
1.2) Utilisation de données 21
1.2.1) Types de décision 22
1.2.2) Gestion de données 22
1.2.3) Objectifs de gestion de données 23
1.3) Introduction aux structures de données 23
1.4) Programmation et langages informatiques 25
1.4.1) Langage de programmation 26
1.4.2) Un programme 27
1.4.3) Passage de l'algorithme au programme 28

CHAPITRE 1

□□□

Introduction générale

Résumé **: Ce chapitre présente, dans une première partie, une brève introduction à l'algorithmique et définit son utilité. Ensuite, et dans une seconde partie, définit quelles sont les données et comment elles soutiennent le système d'information. Le principe de programmation, les types, les langages, ainsi que le passage d'un algorithme à un programme font le sujet de la dernière partie.**

1.1) Introduction à l'algorithmique

1.1.1) Définition d'un algorithme

Un algorithme est un assemblage ordonné d'instructions élémentaires permettant de résoudre un problème de calcul bien spécifié. Autrement, un algorithme contient uniquement des instructions compréhensibles par celui qui devra l'exécuter. Ces instructions permettent de spécifier, en termes généraux, la relation désirée entre un état initial bien spécifié et un état final qui représente le résultat cherché. Les étapes élémentaires sont

éventuellement répétées (instructions de boucle) et sont soumises à des tests logiques (instructions de contrôles). Un algorithme est dit correct si, pour chaque instance en entrée, il se termine en produisant la bonne sortie. Un algorithme peut être spécifié en langage humain ou en langage machine et décrit par les caractéristiques suivantes :

- Les opérations sont toutes définies et portent sur des objets appelés informations,
- L'ordre d'exécution des opérations est défini sans ambiguïté,
- La résolution d'un problème s'effectue d'une manière certaine

Par exemple, pour trier une suite de nombres dans l'ordre croissant. L'entrée est représentée par la suite de n nombres $\{a_1, a_2,..., a_n\}$ et la sortie consiste à réorganiser la suite en entrée de façon qu'on obtiendra la relation $a'_1 \leq a'_2 \leq ... \leq a'_n$ et $\{a'_1,...,a'_n\}$ est inclus dans $\{a_1,..., an\}$.

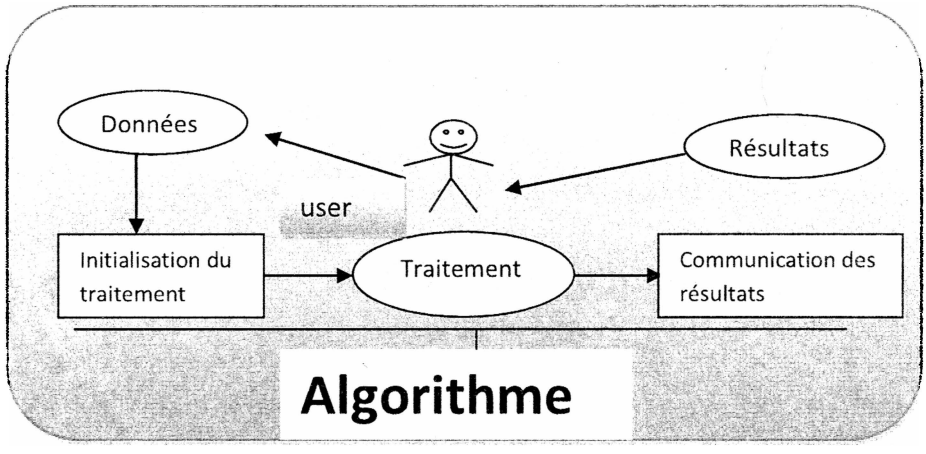


Figure 1. Communication de l'information entre un utilisateur et un algorithme

1.1.2) Rédaction d'un algorithme à partir d'un problème

Il y a 5 étapes à suivre pour arriver à un algorithme à partir d'un problème à savoir :

1. Comprendre le problème.
2. Identifier les données d'entrée et de sortie liées au problème.
3. Formuler le problème en actions élémentaires précises avec les données nécessaires pour le calcul.
4. Développer l'algorithme général suivant les actions.
5. Détailler chaque action de l'algorithme (sans ambiguïté).

🔒 **Exercice :** Nous illustrons par un exemple, comment résoudre un problème permettant de calculer et d'afficher l'aire d'un triangle dont il faut saisir les longueurs des trois côtés à travers le clavier.

🔓 **Solution** : La résolution de ce problème est basée sur la formule : $S^2 = Px (P-A) x (P-B) x (P-C)$ où A, B, C sont les longueurs des trois côtés et P le demi-périmètre du triangle.

➡ **Etape 1** : détermine ce qui est demandé comme résultat (dans notre cas c'est l'aire d'un triangle AT) ;

➡ **Etape 2** : dégager le(s) données d'entrées (dans notre cas les trois côtés A, B, C), le(s) données de sorties (AT) et le(s) données intermédiaires (P qui stocke la valeur du demi périmètre du triangle) ;

➡ **Etape 3** : Les actions élémentaires :

- Déclarer les objets dégagés dans l'étape 2 (A, B, C, T, AT)
- Saisir les trois côtés A, B et C
- Calculer le demi-périmètre P du triangle
- Calculer l'aire AT du triangle.

- Afficher l'aire AT

➡ **Etape 4** :

- Déclarer (A, B, C, P, AT)
- Saisir (A, B, C)
- P=A+B+C/2
- AT= Racine (P.(P-A).(P-B).(P-C))
- Afficher (AT)

➡ **Etape 5** : Rédiger l'algorithme selon les conventions du langage de programmation choisi (l'instruction Lire (A, B, C) est convenable en cas de l'adoption du langage C mais, en cas du langage Pascal, il faut la remplacer par Lire(A) ; Lire(B) et Lire(C)).

Premier algorithme

```
Variables
 Réel A, B, C, P, AT
Ecrire ("Saisir A et B")
 Lire (A, B)
 P= (A+B+C)/2
 AT=(P*(P-A)*(P-B)*(P-C))^0.5
 Ecrire ("L'aire du triangle est =", AT)
```

1.1.3) Efficacité d'un algorithme

La question qui se pose est la suivante : qu'est-ce qu'un bon algorithme ? Un bon algorithme est celui qui répond correctement au problème posé dans un temps rapide et qui n'utilise pas trop de mémoire.

Pour évaluer le temps d'exécution d'un algorithme (un tri) sur une donnée *d* (un ensemble d'entiers à trier), on calcule le nombre d'opérations élémentaires (opérations arithmétiques,

affectation, instruction de contrôle, *etc.*) effectuées pour traiter la donnée *d*.

Au lieu de calculer le nombre d'opérations associées à chaque donnée, on cherche souvent à évaluer le nombre d'opérations nécessaires pour traiter les données qui ont une certaine "taille". Il existe souvent un paramètre naturel qui est un estimateur raisonnable de la taille d'une donnée (par exemple, le nombre n d'éléments à trier pour un algorithme de tri, la taille n, m d'une matrice pour le calcul du déterminant, la longueur n d'une liste pour une inversion, *etc.*).

A titre d'exemple, dans le cas d'un algorithme de calcul de factorielle qui contient n-1 opérations au sein desquelles le nombre d'opérations élémentaires est constant, la complexité est de l'ordre n ($O(n)$).

1.1.4) Utilité d'un algorithme

Un algorithme est une étape primordiale avant la programmation puisqu'il exprime les instructions résolvant un problème donné indépendamment des particularités de tel ou tel langage. L'algorithme est le plan où la rédaction d'une solution, spécifiée par l'orthographe et la grammaire, est compréhensible par son créateur. Donc, apprendre l'algorithme, c'est apprendre à manier la structure logique d'un programme informatique qui se présente dans tout langage de programmation. Cependant, lorsque la rédaction s'effectue sous format d'un programme dans un langage (en C, en Visual Basic, etc.), il faut tout d'abord colleter les problèmes de syntaxe, ou les types d'instructions, propres à ce langage. Et par la suite, apprendre l'algorithme de

manière séparée permet d'éviter la difficulté de traduction directe d'un programme écrit dans un langage bien donné vers un autre programme écrit dans un nouveau langage.

Dans ce livre, nous utilisons une série de conventions appelée «pseudo-code», qui ressemble au langage de programmation authentique dont on aurait évacué la plupart des problèmes de syntaxe (langage C). Ce pseudo-code est susceptible de varier légèrement d'un livre à un autre.

1.1.5) Sous-algorithmes

Un sous-algorithme est conçu pour faire un traitement bien **défini**, bien **délimité**, si possible indépendamment du contexte particulier de l'algorithme appelant. Un sous-algorithme peut appeler un autre et peut être appelé par un algorithme.

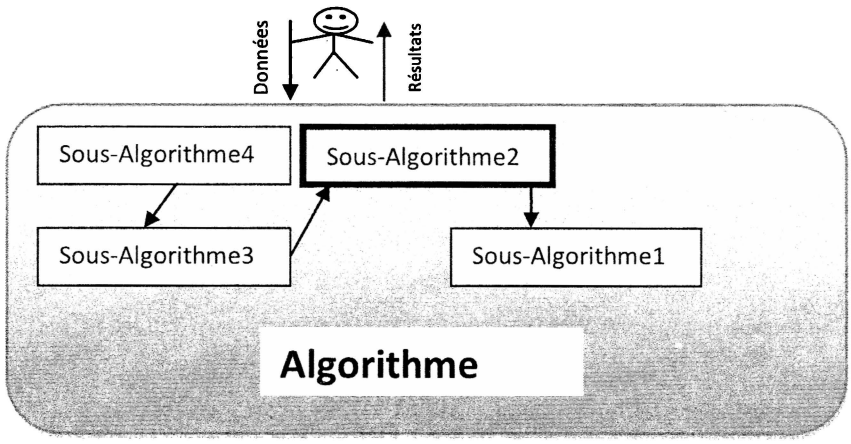


Figure 2. Les éventuels appels entre les sous-algorithmes d'un problème

1.2) Utilisation de données

Avant de discuter les structures de données, il est important de poser des questions de ce genre : Quels sont les données ? Et comment elles soutiennent des systèmes d'information ? La figure suivante montre le rôle qui peut jouer les données dans les événements de n'importe quelle organisation.

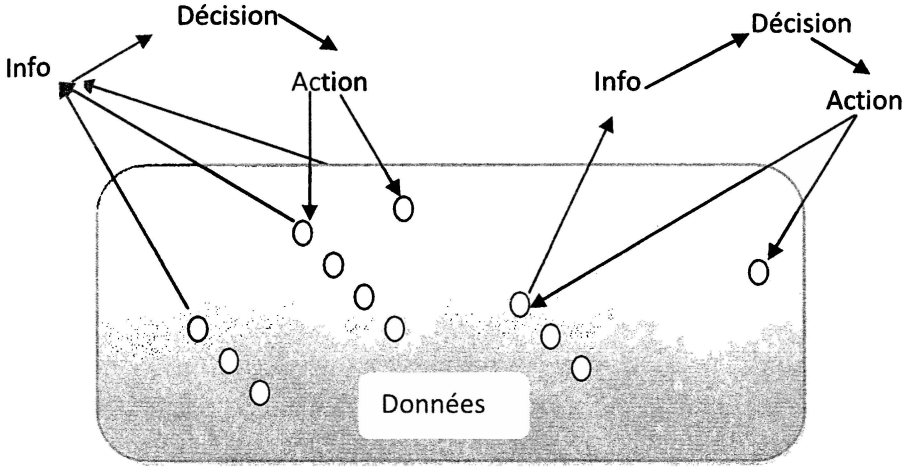


Figure 3. Les données : les ingrédients essentielles pour prendre des actions basées sur la prise de décision informatisée

Le rectangle à coins arrondis représente la source globale de données d'une organisation. Cette source est un participant actif aux opérations et à la planification de l'organisation. Les petits cercles représentent différents éléments ou pièces de données. Ces éléments peuvent être considérés des faits triviaux. Ils sont agrégés et récapitulés dans diverses manières significatives pour constituer l'information. Des décisions sont prises sur la

base de cette information. Les résultats des décisions sont des actions (instructions), qui produisent à leurs tours plus de données. Ces données alors peuvent être incorporées à un autre cycle du processus décisionnel.

1.2.1) Types de décision

Les sources de données doivent soutenir plusieurs types de prise de décision. On peut distinguer trois niveaux de décisions :

- Décisions opérationnelles : qui régissent les activités quotidiennes de l'organisation.
- Décisions de contrôles : qui déterminent la façon avec laquelle l'organisation exécute sa mission rattachée : qui s'appellent les décisions tactiques.
- Décisions de planification : qui développent et définissent la mission de l'organisation. Ces décisions se nomment des décisions stratégiques.

1.2.2) Gestion de données

Les données sont onéreuses. Elles doivent être contrôlées de telle manière qu'elles soient correctes et soient disponibles pour produire l'information nécessaire. Les aspects de manipulation de données incluent ce qui suit :

- Mesure
- Collection
- Transcription

- Validation
- Organisation
- Stockage
- Agrégation
- Mise à jour
- Récupération
- Protection

1.2.3) Objectifs de gestion de données

L'objectif d'un système de gestion de données est de rendre la source de données flexible, et adaptable pour soutenir les processus décisionnels d'une organisation. Quatre directives utiles pour un système de gestion des données, à savoir :

- Des données doivent être représentées et stockées afin de faciliter leur accès plus tard.
- Des données doivent être organisées afin de simplifier leur sélection.
- Des données doivent être traitées et présentées afin de soutenir effectivement l'environnement utilisateur.
- Des données doivent être protégées et contrôlées de sorte qu'elles gardent leurs valeurs.

1.3) Introduction aux structures de données

La connaissance des structures de données est nécessaire pour des personnes qui conçoivent et développent des programmes informatiques de genre : logiciel système ou logiciel d'applica-

tions. Comme cité plus haut les données sont représentées par des valeurs de données apparues temporairement dans le domaine des données du programme et sont enregistrées de manière permanente sur un dossier. Souvent, les différentes valeurs de données sont liées entre elles. Pour permettre à des programmes de se servir de ces relations, ces valeurs de données doivent être sous un format donné. La collection de données organisée s'appelle une structure de données. Les programmes doivent suivre certaines règles pour accéder et traiter les données structurées. Nous pouvons, donc, dire que les données sont représentées comme suit :

Structure de données = Données organisées + opérations allouées.

Une structure de données est une extension du concept du type de données. Les types de données se définissent comme suit :

Type de données = opérations + valeurs de données autorisées.

De plus, nous rappelons qu'un type de données simple peut être employé pour constituer d'autres nouveaux types de données scalaires, par exemple le type énuméré en langage C. De même, il y a des structures de données standard qui sont employées souvent dans un contexte bien déterminé et qui peuvent former la base des structures de données complexes (tableaux).

1.4) Programmation et langages informatiques

Les langages se classifient en deux catégories :

- **Langages naturels** : Ces langages sont compréhensibles par les humains, mais sont ambigües et complexes par rapport à la compréhension d'un ordinateur.
- **Langages de programmation** : Ce sont des outils permettant de décrire des programmes. Ces programmes contiennent des instructions intermédiaires entre le binaire et le langage humain. Chaque langage de programmation possède une sémantique, une syntaxe et des règles de programmation.

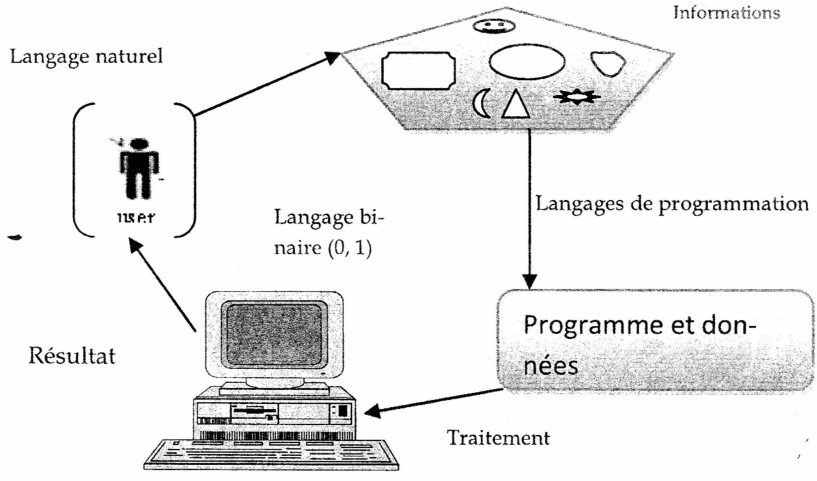


Figure 4. De l'information aux données binaires

1.4.1) Langage de programmation

Un langage de programmation est un langage qui permet de décrire les opérations et leur enchaînement à l'aide des règles d'écritures strictes (une syntaxe), et un sens défini sans ambiguïté (une sémantique) qui se rapproche autant que possible de notre langage naturel et des règles que nous pratiquons couramment pour l'écriture d'expressions arithmétique. C'est le processeur qui se charge d'effectuer les opérations arithmétiques et logiques qui transformeront les données en résultats. Nous pouvons distinguer deux types de langages :

a) langages de bas niveau

Ces langages sont proches des langages machines dont les programmes sont écrits par les utilisateurs. Jusqu'à l'année 1945, il n'existe que le langage binaire. À partir de 1950 il est apparu le superbe langage assembleur bien connu pour la programmation machine.

b) langages de haut niveau (langages évolués)

Ces langages sont conçus pour être proches des langages naturels. Depuis l'année 1955, diverses approches de programmation ont été inventées :

- *Programmation procédurale ou algorithmique* : Le principe de cette programmation nécessite la question comment qui représente la traduction d'une démarche hiérarchique : le problème de départ est divisé progressivement en sous problèmes jusqu'à l'arrivé à des problèmes

faciles à résoudre (exemple : langages Fortran, Cobol, Basic, Pascal, C, Ada...).

- *Programmation orientée objet :* un problème est connu d'une manière totalement différente à d'autres méthodes. En effet, en utilisant cette méthode de programmation on regarde quels sont les acteurs qui interagissent dans le problème. À titre d'exemple, dans un problème de gestion de facturation on a l'acteur "client", l'acteur "facture" et l'acteur "entreprise". En programmation orientée objet, les acteurs sont appelés des objets (exemple : langage SmallTalk, C++, Delphi, Java...).
- *Programmation logique* : La caractéristique principale de cette programmation réside dans la délégation de la charge du comment au langage lui-même. La programmation doit donc se contenter de définir les faits et les règles utilisées pour résoudre un problème (exemple : langage Prolog...).
- *Programmation fonctionnelle* : Ce type de programmation permet une description du résultat comme une composition des fonctions agissant sur des objets (exemple : langage Lisp...).

1.4.2) Un programme

Un programme informatique (application, logiciel) est décrit par un ensemble de données (données d'entrée) et un ensemble d'actions opérant sur ces données (un algorithme) afin de produire un (des) résultat(s) (données de sortie). Pour être compréhensible par un ordinateur (une machine), un programme doit être écrit dans un langage de programmation. L'objectif d'un programme est d'utiliser l'ordinateur pour traiter des données afin d'obtenir des résultats. Pour être utilisable, un ordinateur doit disposer d'un ensemble de programmes prédéfinis qui désigne le logiciel de base. Le rôle de ce logiciel

est de permettre le développement et l'exécution des programmes conçus par les utilisateurs.

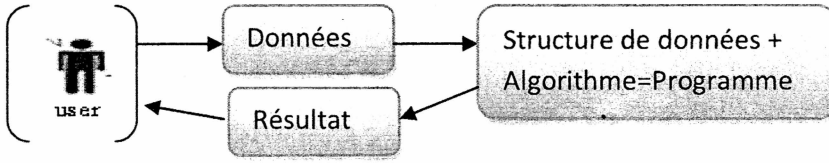


Figure 5. Exécution d'un programme

1.4.3) Passage de l'algorithme au programme

Il s'agit de traduire l'algorithme avec un langage de programmation :

- En respectant la grammaire du langage (syntaxe)
- En utilisant des mots réservés pour le langage
- En maîtrisant des opérateurs et des types de données définis dans le langage

☐ **Exemple du langage C :**

- Mots réservés : strlen, scanf, printf, while, if, else, struct...
- Opérateurs : <, >, =, /, *, ==, &&, | | ...,
- Types de donnée : float, char, int, void,...

Chapitre 2

Notions de base

2.1) Formalisme d'un algorithme **31**

2.2) La notion d'objet **32**

2.2.1) Définition d'une variable **33**

2.2.2) Définition d'une constante **33**

2.3) Types de données **34**

2.3.1) Les types scalaires **35**

2.3.2) Les types élémentaires **39**

CHAPITRE 2

□□□

Notions de base

Résumé : Avant de commencer à apprendre les algorithmes, il est important de connaître les fondements de base en ce qui concerne la structure de rédaction d'un algorithme, les différents types d'objets à utiliser dans un algorithme, ainsi que la codification des objets en mémoire de l'ordinateur. En plus, il est important de connaître les différents types d'instruction/actions contenus dans un algorithme. Ce chapitre se préoccupe par l'éclaircissement de tous ses fondements.

2.1) Formalisme d'un algorithme

Un algorithme doit être lisible et compréhensible par plusieurs personnes, il doit donc suivre des règles. Un algorithme est composé d'un entête et d'un corps comme décrit dans les points ci-dessous :

Entête d'un algorithme : **L'entête** d'un algorithme doit contenir les éléments suivants :

- Les données "en entrée" : c'est-à-dire les éléments qui sont indispensables à son bon fonctionnement (Données d'entrées ou **D.E**) ;
- Les données "en sortie" : c'est-à-dire les éléments calculés, produits, par l'algorithme (Données de sortie ou **D.S**)
- Les données locales à l'algorithme qui lui sont indispensable (données intermédiaires ou **D.I**).

Corps d'un algorithme : Le corps d'un algorithme est composé des éléments suivants :

- du mot clef **début**
- d'une suite d'instructions **indentées** (Instructions élémentaires, instructions de test (conditionnel), instructions de boucles)
- du mot clef **fin**

Exemple d'un algorithme complet :

```
Variables
D.E : Entier a, b
D.S : Entier c
Début
C← a+b
Fin
```

Exemple d'un algorithme complet

2.2) La notion d'objet

L'ordinateur traite les données et les résultats en tant qu'objets. Chaque objet est stocké dans une zone-mémoire dont

la gestion sera prise en charge par l'ordinateur. Un objet peut être une valeur de variable ou une constante.
D'une manière analogue à une armoire de boites à lettres qui désigne des boites postales étiquetées : l'armoire serait la mémoire de l'ordinateur, les boites postales seraient les variables (l'étiquette (numéro de boite) correspondrait au nom de la variable), le code de la poste serait l'adresse mémoire de la variable. Le contenu d'une boite serait la valeur de la variable correspondante, la couleur de la boite serait le type de la variable (bleu pour les particuliers, rouge pour les sociétés, vert pour les établissements, etc.).

2.2.1) Définition d'une variable

Une variable est une entité qui contient une information. L'information contenue dans une variable peut être modifiée au niveau du corps d'un algorithme. Une variable est caractérisée par un nom (identifiant) qui sert à la désigner et un type qui caractérise l'ensemble des valeurs qui peuvent être prises par la variable. L'ensemble des variables est stocké dans la mémoire de l'ordinateur.

2.2.2) Définition d'une constante

Une constante est un objet de valeur constante. Elle représente la réalisation d'une valeur d'un type particulier. D'une manière analogue à une variable, la constante est caractérisée par un nom et un type. Par contre à une variable, la constante ne devrait jamais changer de valeur dans le corps d'un algorithme.

2.3) Types de données

Il ne suffit pas de créer une variable (réserver un emplacement-mémoire). Il faut, encore, préciser ce qu'il doit contenir, car la taille de la variable (taille de l'emplacement-mémoire) dépend de cette spécification qui désigne le type de codage utilisé. Le type d'une variable caractérise l'ensemble des valeurs qui peuvent être prises par la variable et l'ensemble des actions qu'on peut y appliquer. Lorsqu'une variable apparaît dans l'entête d'un algorithme, on lui associe un type en utilisant la syntaxe suivante :

- Type nom_Variable

D'une manière analogue à une variable, la déclaration d'une constante, qui apparaît dans l'entête d'un algorithme, se ressemble à la syntaxe suivante (en majuscule):

- Constante type ID_CONSTANTE

☐ **Exemple 1 : déclarer l'âge et le nom d'une personne**

- Entier age
- Chaîne nom

Règles

- Une fois qu'un type de données est associé à une variable, cette variable ne peut, en aucune place, changer de valeur ;
- Une fois qu'un type de données est associé à une variable, le contenu de cette variable doit obligatoirement être du même type.

Dans l'exemple présenté plus haut, la variable «age» est déclarée de type entier. Cependant, cette variable ne pourra pas stocker des réels, ni changer de type.

- **Exemple 2 :** déclarer la constante ∏

- Constante Réel pi=3.14159
- Constante Réel TVA=0.1

Il y a deux grandes catégories de types : les types simples ou scalaires et les types élémentaires. Les types élémentaires incluent les booléens, les caractères, les chaînes de caractères, le type énuméré et le type structuré :

2.3.1) Les types scalaires

Il y a deux grandes catégories d'objets dans les algorithmes : Les entiers en virgule fixe et les réels en virgule flottante.

a) Les entiers

Selon Wikipédia[1], un **entier naturel,** en mathématique, est un nombre positif (ou nul) permettant fondamentalement de dénombrer des objets comptant chacun pour *un.* Un tel nombre entier peut s'écrire avec une suite finie de chiffres, sans signe et sans chiffres « après la virgule ». En informatique, le calcul entier est relativement simple. Les nombres sont en fait calculés

[1] http://wikipedia.fr/

dans une arithmétique modulo $N = 2^n$ où n est le nombre de bits des mots machine.

n	N		Exemple
16	65 536	= 6×10^4	Macintosh SE/30
32	4 294 967 296	= 4×10^9	Pentium
64	18 446 744 073 709 551 616	= 2×10^{19}	Alpha

Tableau 1. Borne supérieure des nombres entiers

En informatique, il faut pouvoir tout de même désigner des nombres négatifs. La notation couramment utilisée est celle du complément à 2. En notation binaire, le bit le plus significatif est le bit de signe. Au lieu de parler de l'arithmétique entre 0 et 2^n - 1, les nombres sont pris entre -2^{n-1} et 2^{n-1} - 1. Soit, pour n=16, sur l'intervalle [-32768, 32767].

Une opération entre 2 nombres peut créer un débordement, c'est-à-dire atteindre les bornes de l'intervalle. Mais elle respecte les règles de l'arithmétique modulo N. Selon le langage de programmation et son implémentation sur une machine donnée, ce débordement est testé ou non, selon le langage.

Les déclarations des entiers			Exemples de déclaration	Codage
Entier	**Entcourt**	Désigne les entiers négatifs et positifs	Entier année EntCourt vitesse EntLong facto-riel	**2 octets**
	Entlong			**8 octets**
Naturel	**NatCourt**	Désigne les entiers positifs	Naturel age	**2 octets**
	NatLong			**4 octets**

Tableau 2. Déclaration variée des entiers

b) Les réels (nombres flottants)

La représentation flottante sert à désigner les nombres réels pour permettre d'obtenir des valeurs irréalisables en notation fixe. Deux parties significatives qui composent un nombre flottant, *la mantisse*, et *l'exposant*. Un nombre flot
tant tient souvent sur le même nombre de bits n qu'un nombre entier. En flottant, on décompose $n = s + p + q$ en trois champs spécifiés par le signe, la mantisse et l'exposant, qui sont donnés par la machine.

signe decimal e exposant ➔ signe decimal $\times 10^{\text{exposant}}$

Paramètre	Simple	Double
p	23	52
Q	8	11
e_{max}	+127	+1023
e_{min}	-126	-1022

Tableau 3. Formats des flottants IEEE

En plus, les nombres flottants sont une approximation des nombres réels, car leur partie significative f ne tient que sur un nombre p de bits fini. Par exemple, $p = 23$ en simple précision. Le nombre de bits pour l'exposant e est $q = 8$ en simple précision, ce qui fait que le nombre de bits total, avec le bit de signe, est bien 32.

Pour rendre des programmes portables en utilisant les nombres flottants, une norme IEEE 754 (*the Institute of Electrical and Electronics Engineers*) a été définie. Les formats en bits simple et double précision sont représentés dans le tableau suivant :

Le type réel comprend les variables numériques qui ont des valeurs réelles. La plage des valeurs du type réel est :

- -3,40x10 ^{38} à -1,40x10 ^{45} pour les valeurs négatives
- 1,40x10 $^{-45}$ à 3,40x10 ^{38} pour les valeurs positives

Les déclarations des réels			Exemple de déclarations	Codage
Réel	**Réel**	Désigne les valeurs contenues dans l'ensemble R.	Réel taille, taux_tva	**4 octets**
	Réel-double	Réel avec une précision double.	Réeldouble MontantNet ;	**8 octets**

Tableau 4. Déclaration des réels

2.3.2) Les types élémentaires

a) Les booléens

Les variables booléennes (boolean en anglais) ne peuvent prendre que deux valeurs possibles : vrai (true) ou faux (false) et sont codées en 0 ou 1. Ces valeurs sont analogues aux états possibles d'un interrupteur : ouvert ou fermé, d'une lampe : allumée ou éteinte, d'un composant électronique du type transistor : conducteur ou isolant. En bref, les booléens ne sont utiles que pour connaître un état : vrai ou faux et en général pour caractériser si une condition est vraie ou fausse. Les booléens peuvent être utilisés sans toujours le savoir dans les blocs **Si** : si telle condition est vraie, alors...

- Exemple de déclaration des booléens

Les variables booléennes ne peuvent prendre que les valeurs VRAI ou FAUX. Ci-dessous quelques exemples de déclarations :

- **Booléen masculin**
- **Booléen premier, existe**

b) Les caractères

Selon le dictionnaire de français, un caractère est un signe écrit ou tracé. Un caractère peut être une lettre de l'alphabet, un signe de ponctuation ou un chiffre. Un caractère se note entre apostrophes : par exemple, 'A' (caractère A), 'a' (caractère a) etc. Les caractères spéciaux peuvent être désignés directement par leur symbole : ENTER (retour chariot), TAB (tabulation), BEEP (son), etc.

- Exemple de déclaration des caractères
- Caractère c

c) Les chaînes de caractères

La chaîne de caractères est une suite de symboles entre guillemets. Le nombre de caractères dans la chaîne de caractères détermine la longueur de la chaîne. Cette suite de caractères est délimitée par apostrophes, simples ou doubles. Par exemple, "cours" ou "algorithmique".

- Exemple de déclaration des chaînes de caractères
- Chaînes ch

d) Les types énumérés

Un type énuméré est un type dont les variables associées n'auront qu'un nombre très limité de valeur (au maximum 256 différentes valeurs possibles). La définition d'un type énuméré

consiste à déclarer une liste de valeurs possibles (256 au maximum) associées à un type, c'est-à-dire qu'une variable de type énuméré aura l'une et une seule de ces valeurs et pas une autre. Les variables ne peuvent prendre que les valeurs explicitées, par exemple les jours de la semaine (du lundi au dimanche)

- Déclaration des énumérés

➡ joursDeLaSemaine jour ; Si l'informaticien veut utiliser des énumérés, il doit définir le type dans l'entête de l'algorithme en explicitant toutes les valeurs de ce type de la façon suivante :

➡ *joursDeLaSemaine = {lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche}*

e) Le type structuré

Avec le type structuré (encore appelé enregistrement), dans une seule variable on peut réussir à placer des sous-variables nommées champs. Les champs sont placés dans un bloc début ... fin ; et un sous-type peut lui-même être de type enregistrement.

- Déclaration d'un type structuré

Type Structure *nom_du_type*
Début
nouveau_type1 sous_type1
sous_type2_1 sous_type2
structure no_du_type1 typ1
Fin

2.4) Les expressions et les opérateurs

Une expression est un objet syntaxique obtenu en assemblant correctement des constantes, variables et opérateurs. Par

exemple x+3, est une expression. En d'autres termes, elle est une combinaison d'opérateur (s) et d'opérande (s), elle est évaluée durant l'exécution de l'algorithme et possède une valeur (son interprétation) et un type.

2.4.1) Les opérateurs

Un **opérateur** est un symbole d'opération qui permet d'agir sur des variables ou de faire des "calculs". Un opérateur peut être unaire ou binaire :

- ✓ **Unaire :** s'il n'admet qu'un seul opérande, par exemple l'opérateur **Non**
- ✓ **Binaire :** s'il admet deux opérandes, par exemple l'opérateur **+**

Un opérateur est associé à un type de donnée et ne peut être utilisé qu'avec des variables, des constantes, ou des expressions de ce type. La signification d'un opérateur peut changer en fonction du type des opérandes. Par exemple, l'opérateur + avec des entiers aura pour sens l'addition, mais avec des chaînes de caractères aura pour sens la **concaténation.**

Règles

- On ne peut pas additionner un entier et un caractère ;
- Toutefois, exceptionnellement, dans certains cas, on accepte d'utiliser un opérateur avec deux opérandes de types différents (le cas avec les types arithmétiques (4+2.5)).

Nous pouvons distinguer plusieurs types d'opérateurs. Le tableau 5 montre une explication détaillée sur les combinaisons possibles de deux opérandes booléens. Les opérateurs booléens sont représentés par les expressions ET, OU et NON.

Type d'opérateur	Exemples		
Opérateurs booléens			
OU	**a**	**b**	**a OU b**
	Vrai	Vrai	Vrai
	Vrai	Faux	Vrai
	Faux	Vrai	Vrai
	Faux	Faux	Faux
OUExclusif	**a**	**b**	**a OUExclusif b**
	Vrai	Vrai	Faux
	Vrai	Faux	Vrai
	Faux	Vrai	Vrai
	Faux	Faux	Faux
ET	a	b	**a ET b**
	Vrai	Vrai	Vrai
	Vrai	Faux	Faux
	Faux	Vrai	Faux
	Faux	Faux	Faux
Non	a		**NON a**
	Vrai		Faux

	Faux	Vrai

Tableau 5 Les opérateurs booléens

Les opérateurs sur les énumérés sont représentés par les opérateurs succ, ord et pred. Le tableau 6 présente un exemple de manipulation de ces opérateurs.

Type d'opérateur	Exemples
Opérateurs sur les énumérés	
permet d'obtenir le successeur d'un élément.	par exemple avec le type JourDeLaSemaine :**succ Lundi** vaut Mardi
permet d'obtenir le prédécesseur d'un élément	par exemple avec le type JourDeLaSemaine :**pred Lundi** vaut Dimanche
permet d'obtenir le naturel de l'énuméré spécifié dans la bijection du type énuméré	par exemple avec le type JourDeLaSemaine :**ord Lundi** vaut 1
	ord Mercredi vaut 3

Tableau 6 Les d'opérateurs énumérés

Le tableau 7 contient des exemples de manipulation des opérateurs sur les caractères, des opérateurs arithmétiques et des opérateurs de comparaison.

Les opérateurs sur les caractères
Pour les caractères on retrouve les trois opérateurs des énumérés avec en plus un quatrième opérateur nommé **car** qui est le dual de l'opérateur **ord** avec comme fonction de bijection la table de correspondance de la norme ASCII.

ord	ord A vaut 65
car	car 65 vaut A
pred	pred A vaut @
Opérateurs arithmétiques	
+	Addition
-	Soustraction
/	Division entière (11/2=5)
*	Multiplication
%	reste de division entière (11%5=1)
div	Division réelle (11 div 2=5.5)
Les opérateurs relationnels (de comparaison)	
=	C'est l'opérateur que l'on retrouve chez tous les types simples et qui permet de savoir si deux opérandes sont égaux.
≤≥, >, <	Adoptés pour les types possédant un ordre entre les opérateurs de comparaison.
<>	Opérateur d'inégalité

Tableau 7 Les d'opérateurs en algorithme et son équivalence en C

2.4.2) Priorité des opérateurs

La priorité des opérateurs lors de l'évaluation d'une expression arithmétique est la suivante (du plus prioritaire au moins prioritaire) :

- [fonctions], [( )],[*, /, %], [+, -]

Tout comme en arithmétique, les opérateurs ont des priorités. Par exemple * et / sont prioritaires sur + et -.

Règles

- Les opérateurs d'un même bloc [ ] ont la même priorité ;
- Pour les expressions booléennes, la priorité des opérateurs est comme suit : non, et, ouExclusif, ou ;
- L'ordre de priorité des opérateurs relationnels adoptés dans une expression relationnelle est le suivant : >, >=, <, <=, ==, !=

5) Les instructions élémentaires

Les instructions indiquent à l'ordinateur ce qu'il doit faire. Puisque la puissance d'une seule instruction est faible, un programme sera en général composé de nombreuses instructions. Les instructions élémentaires apparaissent clairement lorsqu'on considère la structure d'un micro-ordinateur et de ses périphériques. Dans un algorithme, on peut distinguer les instructions :

1. de lecture d'informations provenant du clavier (entrée)
2. de traitement d'informations
3. d'écriture d'informations à l'écran (sortie)
4. lecture d'informations provenant d'un fichier disque
5. écriture d'informations dans un fichier disque

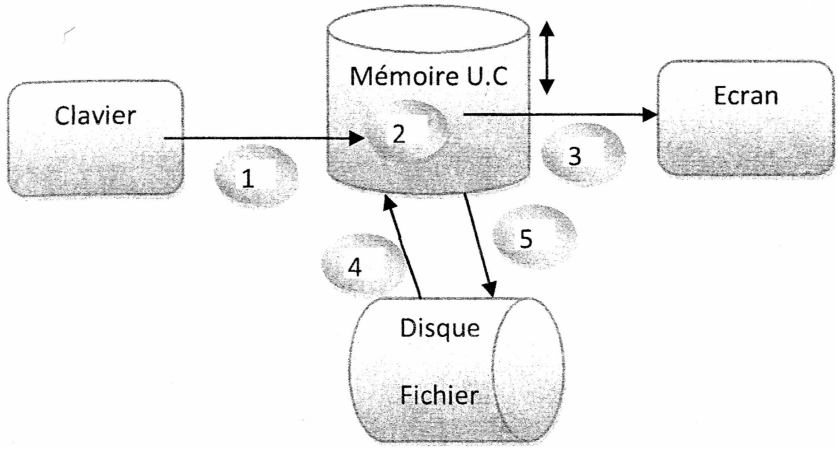


Figure 6. Schéma de communication des informations entre l'ordinateur et ses périphériques

2.5.1) Instruction/expression d'affectation

Pour utiliser une variable dans une expression, il est nécessaire de lui attribuer une valeur. Cette opération s'appelle l'affectation, qui consiste à ranger une valeur (le résultat d'une expression) dans l'emplacement mémoire occupé par une variable. Nous utiliserons le symbole (←) pour décrire cette opération. D'une manière analogue, l'affectation d'un nouveau contenu dans une variable est similaire à l'emplacement d'une lettre dans une boite à lettres. La syntaxe de cet opérateur est comme suit :

- nom_variable←expression

- **Exemple :** l'expression "c←a + b" se comprend de la façon suivante : On prend la valeur contenue dans la variable a, puis on prend la valeur contenue dans la variable b, on additionne ces deux valeurs. Enfin, le résultat est placé dans la variable c. Si c avait auparavant une valeur, cette dernière est perdue.

Règles

L'opération nom_*variable=expression ;* est elle-même une **expression** dont l'évaluation s'effectue de la façon suivante:

- La partie droite de l'affectation est évaluée : le résultat est une valeur V.
- la valeur V est affectée à la variable.

2.5.2) Instructions d'entrée/sortie

Pour que les traitements effectués par un ordinateur puissent être sauvegardés ou communiqués à l'extérieur, on dispose les fonctions d'entrée/sortie. Un algorithme peut interagir avec un utilisateur tel que l'affichage d'un résultat (du texte ou le contenu d'une variable) et/ou demander à un utilisateur de saisir une information afin de la stocker dans une variable. Pour afficher une information, on utilise l'instruction **écrire** suivie entre parenthèses de la chaîne de caractères entre guillemets et/ou des variables de type simple à afficher séparées par des virgules, par exemple :

- **Ecrire** ("La valeur de la variable est **Entier**", a)

Pour donner la possibilité à l'utilisateur de saisir une information, on utilise l'instruction **Lire** suivie entre parenthèses de la/les variable (s) qui reçoi (t/vent) la/les valeur (s) saisie (s) par l'utilisateur, par exemple :

- **Lire** (b, c)

Remarque :

La fonction Lire attend la saisie (au clavier) d'une valeur (val), et une fois que celle-ci est tapée, puis suivie de la touche "ENTER", la valeur est transmise à la variable.

2.5.3) Exemple d'entrée/sortie

Application : *écrire l'algorithme permettant de convertir un montant en euros vers son équivalent montant en dinar tunisien. Cet algorithme reçoit en entrée une somme en euro et affiche en résultat une somme convertie en dinars.*

Solution :

Variables
D.E : Réel valeurEnEuro
D.S : Réel valeurEnDinar
D.I : Réel tauxConversion
Début
tauxConversion ← 1.89
Ecrire (" Taper une somme en euro :")
Lire (valeurEnEuro)
valeurEnDinar ← valeurEnEuro * tauxConversion
Ecrire ("La somme **Réel** en euros se convertit en la somme **Réel** en Drs", valeurEnEuro, valeurEnDinar)
Fin

Remarque : Dans le reste de ce livre nous utilisons les abréviations suivantes : D.E pour désigner les données d'entrées, D.S pour désigner les données de sorties et D.I pour désigner les données intermédiaires (auxiliaires).

2.5.4) Règles pour la rédaction des instructions d'E/S

Le tableau ci-dessous, montre les règles syntaxiques pour l'utilisation des instructions d'entrées/sorties. Ces règles syntaxiques sont plus proches aux règles de la programmation en

langage C. Cependant, l'instruction écrire ne doit contenir que deux guillemets qui peuvent contenir des messages textuels mélangés par les types de variables qui désignent le format dont on voulait afficher. Alors que l'instruction Lire contient uniquement le nom de la/les variables dont on voulait affecter une valeur saisie au clavier.

Rôle d'instruction	**Instruction en algorithme**
Affichage de la valeur d'un entier x et retour à la ligne	**Ecrire** ("La valeur de x est : **Entier RC**", x) ; //RC : retour en ligne
Affichage d'un message qui demande la saisie d'un entier x	**Ecrire** ("Donner un entier x:") ;
Affichage d'un entier x et d'un réel y et de leur produit p	**Ecrire** ("Le produit de **Entier** par **Réel = Réel**", x, y, p) ;
Lecture d'un entier x	**Lire** (x) ;
Lecture d'un entier x et d'un réel y	**Lire** (x, y) ;

Tableau 8. Syntaxe d'utilisation des fonctions d'entrée/sortie en algorithme

Exercices du chapitre 2

Exercice 2.1

Trouver les valeurs des variables A et B après exécution des instructions suivantes ?

Entier A, B
Début
A← 2
B← A + 5
A ←3
Fin

Exercice 2.2

Trouver les valeurs des variables A, B et C après l'exécution des instructions suivantes ?

Entier A, B, C
Début
A←5
B←10
C←A + B
A←A + B
A←C
Fin

Exercice 2.3

Quel résultat produit l'algorithme suivant ?

Chaines A, B, C
Début
A←"42"

B← "24"
C← A + B
Fin

Exercice 2.4

Quelles seront les valeurs des variables A, B et C après exécution des instructions suivantes :

Entier A, B, C
Début
A←1
B←5
C← A-B
A←2
C ←A+B
Fin

Exercice 2.5

Quel résultat produit l'algorithme suivant ?

Entier Val, Double
Début
Val ← 20
Double ← Val * 2
Ecrire (Val)
Ecrire (Double)
Fin

Exercice 2.6

Écrire un algorithme qui demande deux nombres entiers à l'utilisateur, puis calcule et affiche le somme de ces nombres.

Exercice 2.7

Écrire un algorithme qui demande deux valeurs entières à l'utilisateur, puis effectue la somme et la multiplication de ces deux valeurs dans deux variables intermédiaires (S et M), enfin calcule la moyenne de S et M dans une variable de sortie.

Exercice 2.8

Écrire un algorithme qui demande à un utilisateur de saisir deux valeurs de type réel. Cet algorithme calcule la somme de ces deux réels dans une variable accumulateur et enfin incrémente l'accumulateur par 4 et affiche sa valeur finale.

Exercice 2.9

Écrire un algorithme qui affiche le quotient et le reste de division entière de deux nombres entiers saisi au clavier.

Exercice 2.10

Écrire un algorithme qui permet de convertir une durée en secondes vers sont équivalente durée en heures, minutes et secondes. L'algorithme doit afficher le résultat en sortie comme suit :

Donner une durée en secondes !

4200

4200 secondes admet 1h : 10m : 0s

Exercice 2.11

Écrire un algorithme météo qui permet :

- De Lire une température en Celsius.
- D'effectuer la conversion en Fahrenheit en utilisant la formule : Tf=5/9*(T+32)
- D'afficher le résultat de la conversion.

Exercice 2.12

Écrire un algorithme bulletin qui permet :

- de Lire le nom d'un étudiant
- de Lire 3 notes : TP (20% de la moyenne). Examen partiel (30%), examen final (50%)
- de calculer la moyenne de l'étudiant
- d'imprimer le résultat à l'écran.

Exercice 2.13

Écrire un algorithme qui demande à l'utilisateur de saisir quatre nombres entiers et de calculer leurs sommes :
A) En se servant de 5 variables

B) En se servant de 2 variables.

Exercice 2.14

Écrire un algorithme qui permet d'échanger (permuter) et d'afficher les valeurs de deux entiers naturels.

Corrigé des exercices du Chapitre 2

Exercice 2.1 (Solution)

Après exécution de l'instruction	La valeur des variables A et B est :
A← 2	A=2, B= ?
B← A + 5	A=2, B=7
A ←3	A=3, B=7

Après l'exécution des instructions dans l'exercice 1, A vaut 3 et B vaut 7.

Exercice 2.2 (Solution)

Après exécution de l'instruction	La valeur des variables A, B et C est :
A← 5	A=5, B= ?, C= ?
B← 10	A=5, B=10, C= ?
C ←A+B	A=5, B=10, C= 15
A←A+B	A=15, B=10, C= 15

Après l'exécution des instructions dans l'exercice 2, A vaut 15, B vaut 10 et C vaut 15.

Exercice 2.3 (Solution)

Cet algorithme permet de concaténer deux chaines. A la fin de l'algorithme, C vaudra «4224".

Exercice 2.4 (Solution)

Après exécution de l'instruction	La valeur des variables A et B est :
A←1	A=1, B= ?, C= ?
B←5	A=1, B= 5, C= ?
C← A-B	A=1, B= 5, C= -4
A←2	A=2, B= 5, C= -4
C←A+B	A=2, B= 5, C= 7

A la fin de l'exécution de cet algorithme A vaut 2, B vaut 5 et C vaut 7.

Exercice 2.5 (solution)

Après exécution de l'instruction	La valeur des variables A et B est :
Val ← 20	Val=20, Double= ?
Double ← Val * 2	Val=20, Double= 40
Ecrire (Val)	Affiche 20
Ecrire (Double)	Affiche 40

Cet algorithme permet d'afficher deux valeurs entières 20 et 40.

Exercice 2.6 (Solution)

D.E : Entier A, B
D.S : Entier S
Début
Ecrire (''Donner un entier A et un entier B :")
Lire (A, B)
S←A+B
Ecrire ("La somme d'un entier et d'un entier = un entier", A, B, S)

Fin

Exercice 2.7 (Solution)

D.E : Entier A, B
D.I : Entier S, M
D.S : Entier Moy
Début
Ecrire (''Donner un entier A et un entier B :")
Lire (A, B)
S←A+B
M←A*B
Moy←(S+M) div 2
Ecrire ("La moyenne d'un entier et d'un entier = un réel", S, M, Moy)
Fin

Exercice 2.8 (Solution)

D.E : Réel A, B
D.S : Réel Acc
Début
Ecrire (''Donner deux réels A et B")
Lire (A, B)
Acc←A+B
Acc←Acc+4
Ecrire ("La valeur de l'accumulateur réel est", Acc)
Fin

Exercice 2.9 (Solution)

D.E : Entier A, B
D.S : Réel quot, reste

Début
Ecrire (''Donner deux réels A et B")
Lire (A, B)
quot←A/B
reste←A%B
Ecrire ("La division d'un entier par un entier =un entier et reste un entier", A, B, quot, reste)
Fin

Exercice 2.10 (Solution)

D.E : Entier durée
D.S : Entier nh, nm, ns
Début
Ecrire (''Donner une durée en secondes")
Lire (durée)
nh←durée/3600
nm←(durée%3600)/60
ns←durée%60
Ecrire ("La durée entière en secondes = un entier heur : un entier minutes : un entier secondes", durée, nh, nm, ns)
Fin

Exercice 2.11 (Solution)

D.E : Réel T
D.S : **Réel** Tf
Début
Ecrire ("Donner une température en Celsius")
Lire (T)
Tf←(5 div 9)*(T+32)
Ecrire ("La valeur réelle de la température en Celsius convertie en Fahrenheit =un réel", T, Tf)

Fin

Exercice 2.12 (Solution)

D.E : Chaine nom, Réel tp, ep, ef
D.S : Réel moyenne
Début
Ecrire ("Donner le nom de l'étudiant")
Lire (nom)
Ecrire ("Donner trois notes")
Lire(tp, ep, ef)
moyenne←tp*0.2+ep*0.3+ef*0.5
Ecrire ("La moyenne de l'étudiant de nom **Chaîne** = **Réel**", nom, moyenne)
Fin

Exercice 2.13 (Solution)

A)
D.E : Entier A, B, C, D
D.S : Entier som
Début
Ecrire ("Donner 4 entiers")
Lire (A, B, C, D)
som←A+B+C+D
Ecrire ("La somme de ces 4 nombres est =**Entier**", som)
Fin
B)

D.E : Entier A
D.S : Entier som
Début
Ecrire ("Donner le premier entier")
Lire (A)

som←A
Ecrire ("Donner le deuxième entier")
Lire (A)
som←som+A
Ecrire ("Donner le troisième entier")
Lire (A)
som←som+A
Ecrire ("Donner le quatrième entier")
Lire (A)
som←som+A
Ecrire ("La somme de ces 4 nombres est=**Entier**", som)
Fin

Exemple d'exécution à la main d'un algorithme :

Après exécution des instructions	La valeur des variables A et som est :
Ecrire ("Donner le premier entier") Lire (A) som←A	A=5, som=5
Ecrire ("Donner le deuxième entier") Lire (A) som←som+A	A=2, som=7
Ecrire ("Donner le troisième entier") Lire (A) som←som+A	A=3, som=10
Ecrire ("Donner le quatrième entier") Lire (A) som←som+A	A=10, som=20

Exercice 2.14 (Solution)

D.E/D.S : Entier A, B
D.I : Entier aux
Début
Ecrire ("Donner deux entiers")
Lire (A, B)
aux←A
A←B
B←aux
Ecrire ("Les valeurs de A et B après permutation sont représentés par **Entier, Entier** ", A, B)
Fin

Chapitre 3

Les structures conditionnelles

3.1) Rappels sur la logique booléenne 65
3.1.1) Les conditions simples 66
3.1.2) Les conditions complexes 67
3.2) Les structures de choix 68
3.2.1) Structure de choix simple 68
3.2.2) Structure de choix alternative 70
3.2.3) Structure de choix imbriquée 71

CHAPITRE 3

□□□

Les structures conditionnelles

Résumé : Une structure de test est très importante dans la rédaction des algorithmes résolvant des problèmes incluant des actions alternatives ou des actions exécutant sous l'intervention de la validité ou de l'invalidité d'une telle condition. Dans ce chapitre, nous présentons les différents types de structures de tests, la syntaxe de leurs définitions dans un algorithme, ainsi que leurs conditions d'utilisation.

3.1) Rappels sur la logique booléenne

La logique booléenne admet uniquement deux valeurs possibles : Vrai ou Faux. Les opérateurs logiques adoptés dans une expression booléenne sont : **ET, OU, NON** et optionnellement **ouExclusif** qui combine les opérateurs précédemment cités.

- **A ouExclusif B = (NON A ET B) ou (A ET NON B)**

Règles

- Les opérateurs booléens ont la priorité suivante : NON, ET, OU ;
- Les opérateurs booléens ET, OU sont associatifs : A ET (B ET C) = (A ET B) ET C ;
- Les opérateurs booléens ET, OU sont commutatifs : A ET B = B ET A ; A OU B= B OU A ;
- Les opérateurs ET, OU sont distributifs : A OU (B ET C)= (A OU B) et (A OU C) ; A ET (B OU C)= (A ET B) OU (A ET C) ;
- L'opérateur Non est involutif : NON NON A= A ;
- La loi de Morgan appliquée sur l'opérateur NON est comme suit : NON (A OU B)= NON A ET NON B ; NON (A ET B)= NON A OU NON B ;

3.1.1) Les conditions simples

La comparaison entre deux expressions du même type caractérise une condition simple. Cette comparaison s'effectue avec des opérateurs de comparaison. Voici la liste de ces opérateurs accompagnés de leurs significations dans les cas des types numériques ou chaînes :

Opérateur	*Signification pour les numériques*	*Signification pour les chaines*
=	égal à	égal à
<>	Différent	différent

<	Inférieur	placé avant dans l'ordre alphabétique
>	Supérieur	placé après dans l'ordre alphabétique
<=	inférieur ou égal	placé avant dans l'ordre alphabétique ou égal
>=	supérieur ou égal	placé après dans l'ordre alphabétique ou égal

Tableau 9. Les opérateurs de comparaison

Pour comparer deux chaînes, c'est l'ordre alphabétique qui est adopté dans le cas de la comparaison de deux lettres majuscules ou minuscules. Mais si l'on compare majuscules et minuscules, il faut savoir que les majuscules apparaissent avant les minuscules. Par exemple : "A" < "a".

3.1.2) Les conditions complexes

La formulation des conditions, dans certains problèmes, ne peut pas être exprimée sous format simple. Cependant, la plupart des langages autorisent des conditions formées de plusieurs conditions simples reliées entre elles par ce qu'on appelle des opérateurs logiques. Ces opérateurs sont : **ET, OU** et **NON**.

- Pour que la condition complexe, "*condition1* **ET** *condition2*", soit VRAIE, il faut impérativement que la *condition1* soit VRAIE et que la *condition2* soit VRAIE.

- Pour que la condition "*condition1* **OU** *condition2*" soit VRAIE, il suffit que *condition1* soit VRAIE OU *condition2* soit VRAIE. Il est à noter que cette condition complexe sera VRAIE si *condition1* ET *condition2* soient VRAIES.
- L'opérateur NON inverse une condition : L'expression "**NON** (*condition*)" est VRAIE si condition est fasusse, et FAUX si la condition est VRAIE.

Remarque :
Généralement, les opérateurs logiques peuvent porter sur des conditions simples, que sur des conditions complexes. L'usage de parenthèses permet, dans de telles situations, de régler d'éventuels problèmes de priorité. Par exemple, la condition : (a < 5 **ET** b > 1) **OU** (a > 2 **ET** b > 3) est VRAIE si l'une au moins des conditions entre parenthèses est VRAIE.

3.2) Les structures de choix

Au contraire de l'interprétation séquentielle des instructions d'un algorithme, l'exécution d'un algorithme peut être conditionnée. Par exemple, la résolution d'une équation du second degré est conditionnée par le signe Δ. L'instruction *si alors sinon* permet de conditionner l'exécution d'un algorithme à la valeur d'une expression booléenne. Cependant, la conditionnelle est une structure de contrôle qui permet de n'effectuer les calculs que si certaines conditions sont vérifiées. Nous pouvons distinguer trois types de structures de contrôle :

3.2.1) Structure de choix simple

La forme simple des structures de choix correspond au cas où la clause sinon serait absente. La syntaxe de cette structure est comme suit :

Si (condition=vrai)
Alors
actions_condition_vraie
Finsi

Structure à choix

- **Application 1 :** *Ecrire un algorithme qui permet de calculer et d'afficher le maximum de deux entiers e1 et e2.*

Solution :

D.E : Entier e1, e2
D.S : Entier maxim
Début
Ecrire ("Saisir e1 et e2 :")
Lire (e1, e2)
maxim ← e1
Si (e2>e1)
Alors
Maxim← e2
Finsi
Ecrire ("Le maximum de **Entier** et **Entier** est = **Entier**", e1, e2, maxim)
Fin

- **Application 2 :** *Ecrire un algorithme qui applique une remise de 1% si le montant dépasse 5000 Drs lors du calcul du montant d'une facture.*

- **Solution :**

D.E/ D.S : Réel montant
Début
Ecrire ("Donner montant :")
Lire (montant)
Si (montant>5000)

Alors
montant←montant*0.99
Finsi
Fin

3.2.2) Structure de choix alternative

La forme alternative ou complète se produit lorsque les deux branches de la structure si sont présentes (présence de sinon). La syntaxe de cette forme se définit comme suit :

```
Si (condition=vrai) Alors
  Débutalors
   actions_condition_vraie
   Finalors
  Sinon
  Débutsinon
   actions_condition_fausse
  Finsinon
Finsi
```

Syntaxe de la structure de choix alternative

Le fonctionnement de cette structure s'explique comme suit : Si la condition a pour valeur VRAIE, la machine exécute la série d'instructions **actions_condition_vraie**. À la fin de cette série d'instructions, au moment où elle arrive au mot "**Sinon**", la machine sautera directement à la première instruction située après le "**Finsi** ". De même, au cas où le booléen aurait comme valeur "**Faux**", la machine sautera directement à la première ligne située après le "**Sinon**" et exécute l'ensemble des instructions **actions_condition_fausse**.

Application : *Écrire un algorithme qui permet de calculer et d'afficher la valeur absolue d'un entier n saisi au clavier*

Solution :

D.E : Entier n
D.S : Entier abs

```
Début
Ecrire ("Saisir un entier n :")
Lire(n)
Si (n≥0)
Alors
abs← n
Sinon
abs← -n
Finsi
Ecrire ("La valeur absolue de Entier est = Entier", n, abs)
Fin
```

3.2.3) Structure de choix imbriquée

Les structures de choix peuvent être imbriquées dans le cas où le problème abordé propose diverses alternatives ou décisions. Il peut arriver que l'une des parties d'une structure alternative contienne à son tour une structure alternative. Dans ce cas, on dit qu'on a des structures alternatives imbriquées les unes dans les autres. La syntaxe de la forme alternative peut être variée. Pour des raisons de simplicité, nous présentons un exemple simple de la syntaxe d'une structure alternative :

Syntaxe de la structure imbriquée

```
Si (condition1=vrai) Alors
  Si (condition2=vrai) Alors
   Débutalors
     actions_condition2_vraie
   Finalors
  Sinon
  Débutsinon //cas d'un bloc d'instructions
  actions_condition2_fausse
  FinSinon
 Finsi
 Sinon //condition1
   actions_condition1_fausse
  Finsi
```

Note : Les mots clés **Débutalors, Finalors, Débutsinon** et **Finsinon** après les écritures «alors» et «sinon» sont placés uniquement dans le cas ou les actions présentent des blocs d'instructions > 1 (plus qu'une instruction).

Exercice : *Ecrire un algorithme qui permet de donner l'état de l'eau selon sa température.*

Solution :

```
DE : Réel t
DS : Chaine etat
Début
Ecrire ("Saisir une température en Celsius :")
Lire(t)
Si (t≤0) Alors
etat← "solide"
```

```
Sinon
Si (t<100) Alors
Etat←"liquide"
Sinon
Etat←"gazeuse"
Finsi
Finsi
Ecrire ("L'état de l'eau de température Réel est =Chaîne",
t, etat)
Fin
```

3.3) Les structures d'aiguillage (de cas)

Lorsque les conditions expriment une égalité stricte (comparer une même variable avec plusieurs valeurs), on peut utiliser une autre syntaxe qui permet de décrire les cas un par un (structure des cas). Cette structure de contrôle qui représente une sorte d'aiguillage, permet de grouper plusieurs cas entre eux, et dispose d'un cas (optionnel) regroupant tous ceux qui ne sont pas décrits explicitement. Cette instruction permet de choisir la séquence d'instructions correspondantes à un cas parmi n. La syntaxe de la structure selon est définie comme suit :

```
Selon (expr )
Faire
cas v1 : action1
cas v21, v22, . . ., v2m : action2
cas v31, . . ., v32 : action3
. . .
cas vn : actionn
sinon : action
FinSelon
```

Syntaxe de la structure des cas

Application : *Ecrire un algorithme qui permet de calculer et d'afficher le nombre de jours correspondants à une date donnée (une année et un mois).*

Solution :

```
D.E : Entier a, m
D.S : Entier nbj
Début
 Ecrire("Saisir l'année a et le mois m:")
 Lire(a,m)
 Selon (m)
  Faire
   cas 1, 3, 5, 7, 8, 10 : nb←31
   cas 2: Si ((a%4=0 et a%100<>0) ou (a%400=0))
        Alors
        nbj←29 Sinon nbj←28
        finsi
   cas 4, 6, 9,11 : nbj←30
   Sinon : nbj←0
 Finselon
 Si (nbj<>0) Alors
 Ecrire("le mois Entier de l'année Entier admet Entier
jours", m, a, nbj)
 Sinon
 Ecrire("date invalide")
 Finsi
Fin
```

Exercices du chapitre 3

Exercice 3.1

Soient x, y, z, t quatre variables numériques d'un environnement donné. Chacune des questions suivantes caractérise une situation de cet environnement. Chaque situation représente une expression booléenne ou conditionnelle. Exprimer ces expressions en utilisant les opérateurs ET, OU et NON.

1. Les valeurs de x et de y sont toutes les deux supérieures à 3
2. Les variables x, y et z sont identiques
3. Les valeurs de x, y et z sont identiques mais différentes de celle de t
4. La valeur de x est strictement comprise entre les valeurs des variables y et t
5. La valeur de x est strictement comprise entre les valeurs des variables y et t et la valeur de y est inférieure à celle de la valeur de t
6. Parmi les valeurs de x, y et z, deux valeurs au moins sont identiques
7. Parmi les valeurs de x, y et z, deux valeurs et seulement deux sont identiques
8. Parmi les valeurs de x, y et z, deux valeurs au plus sont identiques

Exercice 3.2

Soit la suite d'instructions ci-dessous. Remplir le tableau en bas par les actions appropriées.

```
Si c1
 Alors
  a
 Sinon
 DebSinon
  b
  Si c2
  Alors
   c
  Sinon
   D
  Finsi
 Finsinon
Finsi
```

Cond1	Cond2	Actions exécutées

Exercice 3.3

Écrire un algorithme qui demande à l'utilisateur de saisir deux nombres entiers et lui affiche, ensuite, si leur produit est négatif

ou positif (on laisse de côté le cas où le produit est nul). Il ne faut pas calculer le produit de ces deux nombres.

Exercice 3.4

Écrire un algorithme qui demande à l'utilisateur de saisir 3 noms et lui affiche ensuite s'ils sont rangés ou non dans l'ordre alphabétique.

Exercice 3.5

Écrire un algorithme qui demande à l'utilisateur de saisir deux nombres entiers et lui affiche la valeur maximale parmi ces nombres. En cas d'égalité, l'algorithme affiche que les deux nombres sont égaux.

Exercice 3.6

a) Écrire un algorithme qui demande à l'utilisateur de saisir un nombre entier et affiche si le nombre est pair ou impair.
b) Généraliser pour m et n, m est-il un multiple entier de n ?

Exercice 3.7

Écrire un algorithme qui demande à l'utilisateur de saisir deux nombres entiers et lui affiche la somme s'ils sont tous les deux positifs et le produit s'ils sont tous les deux négatifs.

Exercice 3.8

Écrire un algorithme qui demande à un utilisateur de saisir un réel x et affiche, ensuite, la partie du domaine D = [-10,5] à la quelle appartient la valeur de x. L'algorithme doit choisir entre 3 possibilités (x est intérieur, x est extérieur, x est frontière).

Exercice 3.9

Écrire un algorithme qui demande l'âge d'un enfant à l'utilisateur. Ensuite, il lui affiche sa catégorie :

- "Poussin" de 6 à 7 ans
- "Pupille" de 8 à 9 ans
- "Minime" de 10 à 11 ans
- "Cadet" après 12 ans

Exercice 3.10

Écrire un algorithme qui demande un entier positif N ($N \leq 999$). Si $N \leq 999$, l'algorithme affichera les chiffres des unités, dizaines et centaines éventuels, autrement l'algorithme affichera que le nombre est trop grand.

Exercice 3.11

Écrire un algorithme qui demande la saisie d'un montant, puis détermine un montant net par application d'une remise de :

- 1% si le montant est compris entre 2000 et 5000 DT (valeurs comprises)
- 2% si le montant est supérieur à 5000 Dhs.

Exercice 3.12

Écrire un algorithme qui calcule les solutions réelles d'une équation du second degré $ax^2+bx+c = 0$ en discutant la formule :

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Utilisez une variable d'aide **D** pour la valeur du discriminant b^2-4ac et décidez à l'aide de **D**, si l'équation a une, deux ou aucune solution réelle. Considérez aussi les cas ou l'utilisateur entre des valeurs nulles pour a ; pour a et b ; pour a, b et c. Affichez les résultats et les messages nécessaires sur l'écran.

1) **Cas 1** : Utiliser la structure conditionnelle si ;
2) **Cas2** : Utiliser la structure conditionnelle selon.

Exercice 3.13

Un magasin offre une réduction sur achat dans les conditions suivantes :

- Si le montant d'achat (MA) est inférieur à 350 DT, il n'y a pas de réduction (MR).
- Si le montant d'achat est compris entre 350 DT et 600 DT, le taux de la réduction (TR) est 2%.
- Si le montant d'achat est supérieur à 600 DT, le taux de la réduction est 3%.

Écrire un algorithme qui demande la saisie d'un montant d'achat, puis détermine le prix net à payer (PN).

Indications : PN= MA- MR

MR= MA*TR.

Exercice 3.14

Écrire un algorithme qui permet de déterminer si une année est bissextile ou non.
Indications : Une année est bissextile si elle est divisible par 4 et non par 100 ou divisible par 400.

Exercice 3.15

Un magasin désire automatiser l'opération de calcul des remises accordées à ses clients. Ecrire un algorithme qui permet d'afficher le montant net à payer sachant que :

- Net à payer = Montant total de la facture - Remise

La remise se calcule comme suit

- Pour un montant ≥ 150 DT et une ancienneté≥ 2 ans → Taux=10%
- Pour un montant ≥ 100 DT → Taux=5%
- Pour un montant <100 DT → Taux=2%

Indications : L'ancienneté du client et le montant total de la facture sont des données saisies au clavier.

Exercice 3.16

Dans un problème de facturation, on désigne le prix unitaire en Dinars par P et la quantité par Q. Le montant de la facture est défini par l'expression *P*Q* + le port. Pour une commande de plus de 100 DT, le port est gratuit, sinon il vaut 10% du montant de la commande, avec un minimum de 2 DT.

Indications : Il s'agit de deux cas possibles :

Cas1 : La commande dépasse 100 DT.

Cas2 : La commande n'atteint pas 100 DT. Se décompose en 2 sous-cas : 10% de la commande dépasse 2 DT et 10% de la commande ne dépasse pas 2 DT.

Dans la résolution de notre problème on procède comme suit :

- Calculer le *tarif brut*
- *Si tarif brut > 100 : tarif à payer = tarif brut, pas de frais de port.*
- *Si tarif brut <= 100. On calcule les frais de port = tarif brut * 10%.*
- Si les frais de port sont inférieurs à 2 DT, alors on fixe les frais de port à 2 DT dans le cas contraire, on ne touche pas aux frais de port.

Ecrire un algorithme qui détermine le montant de la facture.

Exercice 3.17

Écrire un algorithme qui permet de calculer le montant de la rétribution des heures supplémentaires d'un employé, sachant que le prix unitaires de l'heure s'obtient à travers le barème suivant :

- Les 39 premières heures sont payées au prix de base
- De la 40^{éme} à la 44^{éme} heure il y a une majoration de 50% du prix de base
- De la 45^{éme} à la 49^{éme} heure il y a une majoration de 75% du prix de base
- Au-delà de la 50^{éme} heure il y a une majoration de 100% du prix de base.

Exercice 3.18

Écrire un algorithme qui demande à l'utilisateur de saisir une heure de départ (exemple : 6h : 40m : 30s) et une heure d'arri-

vée (exemple : 8h : 45m : 20s), puis déterminer le temps écoulé entre ces deux instants.
Indications : Il ne faut pas convertir ni les heures, ni les minutes en secondes.

Exercice 3.19

Écrire un algorithme qui demande à l'utilisateur de saisir une date sous la forme jj, mm, aa et qui vérifie la validité de cette date. Dans le cas ou elle est valide, il l'affiche avec le nom du mois en toutes lettres. Par exemple, 28-09-2010 est une date valide et l'algorithme doit alors afficher : 28 septembre 2010.

Exercice 3.20

Écrire un algorithme calculatrice permettant à un utilisateur de saisir deux entiers et un opérateur arithmétique (+,-, /, %) et lui affichant le résultat de l'opération selon le type de l'opérateur saisi.
Indications : En cas d'erreur, le résultat doit contenir la valeur -1.

Exercice 3.21

Ecrire un algorithme qui effectue des calculs d'aires à la demande. L'algorithme à rédiger ne considère que certaines formes assez élémentaires (disque, carré, ellipse, rectangle). Plus précisément, l'algorithme demande à l'utilisateur de saisir une lettre qui lui indique le type de la figure :

- "D" pour disque
- "C" pour carré
- "E" pour ellipse
- "R" pour rectangle

Puis demande de saisir deux nombres réels représentant les dimensions. En cas de doute sur la nature de la figure, le programme affiche la surface=0.

Indications :

- Surface d'un disque=pi x rayon^2
- Surface d'un rectangle = longueur x largeur
- Surface d'une ellipse= pi x d1 x d2 / 4
- Surface d'un carré = côté^2

Corrigé des exercices du Chapitre 3

Exercice 3.1 (Solution)

1. (x>3 ET y>3)
2. (x=y ET y=z)
3. (x=y ET y=z ET z<>t)
4. ((x>y ET x<t) OU (x>t ET x<y))
5. (x>y ET x<t)
6. (x=y OU x=z OU y=z)
7. ((x=y ET x<>z) OU (x=z ET x<>y) OU (y=z ET y<>x))
8. (x<>y OU x<>z OU y<>z)

Exercice 3.2 (Solution)

Cond1	Cond2	Actions exécutées
Vrai	Vrai	a
Vrai	Faux	a
Faux	Vrai	b, c
Faux	Faux	d, d

Exercice 3.3 (Solution)

D.E : Entier a, b
Début
Ecrire ("Entrez deux nombres : ")
Lire (a, b)
Si ((a > 0 ET b>0) OU (a<0 ET b<0)) **Alors**
Ecrire ("Leur produit est positif")
Sinon

```
 Ecrire ("Leur produit est négatif")
 Finsi
Fin
```

Exercice 3.4 (Solution)

```
D.E : Chaîne a, b, c
Début
Ecrire ("Entrez successivement trois noms : ")
Lire (a, b, c)
Si ( a < b ET b < c) Alors
Ecrire ("Ces noms sont classés alphabétiquement")
Sinon
Ecrire ("Ces noms ne sont pas classés")
Finsi
Fin
```

Exercice 3.5 (Solution)

```
D.E : Entier a, b
Début
 Ecrire ("Entrez deux entiers : ")
  Lire (a, b)
  Si (a >b) Alors
   Ecrire ("La maximum est = Entier", a)
  Sinon
  Si (a<b) Alors
  Ecrire ("La maximum est =Entier", b)
  Sinon
  Ecrire ("Les deux entiers sont égaux")
 Finsi
Fin
```

Exercice 3.6 (Solution)

a)

```
D.E : Entier m
 Début
  Ecrire ("Entrez deux entiers : ")
  Lire (m)
   Si (m%2=0) Alors
    Ecrire ("Entier est paire", m)
    Sinon
    Ecrire ("Entier est impaire", m)
   Finsi
 Fin
```

Exemple d'exécution :

Entrez un entier : 12

12 est paire

b)

```
D.E : Entier m, n
 Début
  Ecrire ("Entrez deux entiers : ")
  Lire (m, n)
   Si (m%n=0) Alors
    Ecrire ("Entier est un multiple entier de Entier", m, n)
    Sinon
    Ecrire ("Entier n'est pas un multiple entier de Entier", m, n)
   Finsi
 Fin
```

Exercice 3.7 (Solution)

```
D.E : Entier a, b
 Début
  Ecrire ("Entrez deux entiers : ")
  Lire (a,b)
```

```
    Si (a<0 ET b<0) Alors
     Ecrire ("le résultat du produit de Entier par Entier est =
Entier", a,b, a*b)
     Sinon
    Si (a>0 ET b>0) Alors
     Ecrire ("le résultat de la somme de Entier et Entier est
=Entier", a,b, a+b)
    Finsi
  Finsi
  Fin
```

Exercice 3.8 (Solution)

```
D.E : Réel x
 Début
  Ecrire ("Entrez un réel : ")
   Lire (x)
    Si (x<-10 OU x>5) Alors
     Ecrire ("le réel est extérieure", x)
     Sinon
     Si (x=-10 OU x=5) Alors
     Ecrire ("Réel est frontière", x)
     Sinon
     Ecrire ("Réel est intérieur", x)
    Finsi
  Finsi
  Fin
```

Exercice 3.9 (Solution)

```
D.E : Entier age
Début
Ecrire ("Entrez l'âge de l'enfant : ")
 Lire (age)
 Si (age >= 12) Alors
```

```
Ecrire ("Catégorie Cadet")
 Sinon Si (age >= 10) Alors
 Ecrire ("Catégorie Minime")
  Sinon Si (age >= 8) Alors
  Ecrire ("Catégorie Pupille")
  Sinon Si (age >= 6) Alors
  Ecrire ("Catégorie Poussin")
  Finsi
  Finsi
 Finsi
Fin
```

Exercice 3.10 (Solution)

```
D.E : Entier N
Début
Ecrire ("Donner un entier N : ")
 Lire (N)
 Si (N<999) Alors
 Débutalors
  Ecrire ("unité=un entier", N%10)
   Ecrire ("dizaine=un entier", (N/10)%10)
   Ecrire ("centaine=un entier", N/100)
   Finalors
  Sinon
  Ecrire ("Le nombre est trop grand")
  Finsi
Fin
```

Exercice 3.11 (Solution)

```
D.E/D.S : Réel montant
D.I : Réel taux
```

```
Début
 Ecrire ("Entrez le montant : ")
  Lire (montant)
  Si (montant < 2000) Alors
  taux← 0
  Sinon
  Si (montant ≤ 5000) Alors
   taux← 1
   Sinon
   taux← 2
  Finsi
 Finsi
 Montant ← montant * (1 - taux div 100)
 Ecrire ("Le montant net est =Réel: ", montant)
 Fin
```

Exercice 3.12 (Solution)

1)

```
D.E : Réel a, b, c
D.I : Réel D
Début
  Ecrire ("Entrez 3 coefficients réels ")
  Lire (a, b, c)
  Si(a=0 ET b=0 ET c=0) Alors
    Ecrire("Solutions dans IR")
    Sinon
     Si(a=0 ET b=0) Alors
     Ecrire("Solution Impossible : Contradiction")
     Sinon
     Si ((a=0)) Alors
     Ecrire("La solution est unique= Réel",-c  div b )
Sinon
```

```
DébutSinon
    D=b*b-4*a*c
 Si(D>0) Alors
Ecrire("Deux solutions possibles : Réel et Réel", (-b-(D)^0.5)
div 2*a,  (-b+ (D)^0.5) div 2*a)
    Sinon
     Si (D=0) Alors
      Ecrire("Solution double: Réel", (-b) div 2*a)
       Sinon
       Ecrire("Solution impossible")
      Finsi
       Finsi
       FinSinon
       Finsi
      Finsi
   Finsi
Fin
 2)

D.E : Réel a, b, c
D.I : Réel D
Début
 Ecrire ("Entrez 3 coefficients réels ")
 Lire (a, b, c)
  Si(a=0 ET b=0 ET c=0) Alors
   Ecrire("Solutions dans IR")
   Sinon
    Si(a=0 ET b=0) Alors
    Ecrire("Solution Impossible : Contradiction")
    Sinon
    Si ((a=0)) Alors
    Ecrire("Une solution réelle=Réel",-c  div b )
```

```
   Sinon
    Débutsinon
     D=b*b-4*a*c
     Selon (D)
     Faire
     cas D>0 : Ecrire("Deux solutions possibles : Réel et Réel",
                (-b-(D)^0.5) div 2*a, (-b+ (D)^0.5) div 2*a)
     cas D=0 : Ecrire("Une solution double: Réel", (-b) div 2*a)
     Sinon : Ecrire("Solution impossible")
     FinSelon
    Finsinon
  Finsi
 Finsi
Fin
```

Exercice 3.13 (Solution)

```
D.E : Réel MA
D.I : Réel TR
D.S : Réel PN
Début
   Ecrire ("Entrer le montant d'achat ")
   Lire (MA)
  Si (MA<=350) Alors
   TR←0
   Sinon
    Si (MA<600) Alors
     TR←0.02
     Sinon
     TR←0.03
    Finsi
```

```
 Finsi
 MR←MA*TR
PN←MA-MR
Ecrire ("Le prix Net est = Réel : ", PN)
Fin
```

Exercice 3.14 (Solution)

```
D.E : Entier A
Début
 Ecrire ("Entrer l'année ")
 Lire (A)
  Si((A%4=0 ET A%100<>0)  OU A%400=0) Alors
   Ecrire ("L'année Réel est bissextile", A) ;
   Sinon
   Ecrire("L'année Réel est non bissextile", A) ;
 Finsi
Fin
```

Exercice 3.15 (Solution)

```
D.E : Réel mtFact,  anc
D.I : Réel remise
D.S : Réel nap
Début
 Ecrire ("Entrer le montant total de la facture ")
  Lire (mtFact)
  Ecrire ("Entrer l'ancienneté ")
   Lire (anc)
  Si ((mtFact>=150) ET (anc>=2)) Alors
```

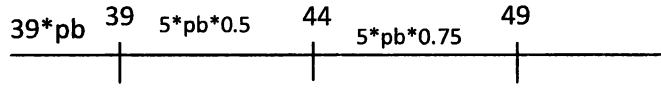
```
    remise←mtFact *0.1
    Sinon
      Si (mtFact>=100) Alors
       remise←mtFact *0.5
      Sinon
      remise←mtFact *0.02
    FinSi
  Finsi
nap← mtFact-remise
Ecrire ("Le prix net à payer est = Réel :", nap)
Fin
```

Exercice 3.16 (Solution)

```
D.E : Réel P, Q
D.I : Réel fport, tb
D.S : Réel montfact
Début
  Ecrire ("Entrer la quantité ")
  Lire (P)
  Ecrire ("Entrer le prixntité ")
   Lire (Q)
   tb←P*Q
  Si(tb>100) Alors
   fport←0
   Sinon
   DébutSinon
   fport←tb*0.1
    Si(fport<2) Alors
     fport←2
    Finsi
```

```
  FinSinon
 Finsi
montfact←tb+fport
Ecrire ("Le montant de la facture est = Réel :", montfact)
Fin
```

Exercice 3.17 (Solution)




```
D.E : Réel pu, nbheur, pbase
D.I : Réel montret
D.S : Réel salaire
Début
 Ecrire ("Entrer le nombre des heures travaillées")
  Lire (nbheur)
  Ecrire ("Entrer le prix de base unitaire d'une heure de travail")
   Lire (pbase)
   Si(tb<=39) Alors
   montret←0
   Sinon
    Si(tb<=44) Alors
     montret←(nbheur-39)*(pb*0.5)
     Sinon
     Si(tb<=49) Alors
      montret←5*(pb*0.5)+(nbheur-44)*(pb*0.75)
       Sinon
      montret←5*(pb*0.5)+ 5*(pb*0.75) +(nbheur-50)*(pb)
     Finsi
    Finsi
 Finsi
```

salaire←pb*39+montret
Ecrire ("La salaire de l'employer est = **Réel** :", salaire)
Fin

Exercice 3.18 (Solution)

D.E : **Entier** hd, ha, md, ma, sd, sa
D.S : **Entier** he, me, se
Début
Ecrire ("Entrer l'heure de départ, ainsi que l'heure d'arrivée ")
Lire (hd, ha)
Ecrire ("Entrer la minute de départ, ainsi que la minute d'arrivée ")
Lire (md, ma)
Ecrire ("Entrer la seconde de départ, ainsi que la seconde d'arrivée ")
Lire (sd, sa)

Si(sa<sd) **Alors**
se←60-se
md←md+1
Finsi
Si(ma<md) **Alors**
me←60-md
hd←hd+1
Finsi
Si(hd=24) **Alors**
hd←0
Finsi
Si(ha<hd) **Alors**
he←hd-ha

Sinon
he←ha-hd
Finsi
Ecrire ("La durée écoulée entre ces deux durées est = **Entier** heures : **Entier** minutes : **Entier** secondes", he, me, se)
Fin

Exercice 3.19 (Solution)

D.E : **Entier** jj, mm, aa
D.S : **Chaîne** nomMois
Début
Ecrire ("Entrer une date sous forme jour, mois, année")
Lire (jj, mm, aa)
Selon (mm)
Faire
cas 1 : nomMois←"Janvier"
cas 2 : nomMois←"Février"
cas 3 : nomMois←"Mars"
cas 4 : nomMois←"Avril"
cas 5 : nomMois←"Mai"
cas 6 : nomMois←"Juin"
cas 7 : nomMois←"Juillet"
cas 8 : nomMois←"Aout"
cas 9 : nomMois←"Septembre"
cas 10 : nomMois←"Octobre"
cas 11 : nomMois←"Novembre"
cas 12 : nomMois←"Décembre"
Sinon : nomMois←" "
FinSelon
Si(nomMois<>" ") **Alors**

Ecrire ("La date s'écrit comme suit : **Entier Chaîne Entier**", jj, nomMois, aa)
Sinon
Ecrire ("Date invalide")
Finsi
Fin

Exercice 3.20 (Solution)

D.E : **Entier** a, b, caractère c
D.S : **Réel** res
Début
Ecrire ("Entrer deux entiers")
Lire (a,b)
Ecrire ("Entrer un opérateur")
Lire (c)
Selon (c)
Faire
cas '+': res←a+b
cas '-': res←a-b
cas '*': res←a*b
cas '/': **Si** (b<>0) **Alors**
res←a div b
Sinon
Res←-1
Finsi
cas '%': **Si** (b<>0) **Alors**
res←a % b
Sinon
Res←-1
Finsi
Sinon : res←-1

```
  FinSelon
   Si(res=-1) Alors
    Ecrire ("opération invalide")
     Sinon
    Ecrire (" Entier Caractère Entier = Réel", a, b, c, res)
   Finsi
Fin
```

Exercice 3.21 (Solution)

```
D.E : Caractère choix
      Réel l, L, c
D.S : Réel aire
Constante Réel PI=3.14
Début
  Ecrire ("Entrer votre choix, D (disque), R (rectangle) , E (ellipse), C (carré)")
  Lire (choix)
  Selon (choix)
   Faire
    cas 'D' : Ecrire ("Entrer le rayon")
              Lire (c)
              aire←PI*c^2
    cas 'R' : Ecrire ("Entrer la largeur")
              Lire (l)
              Ecrire ("Entrer la longueur")
              Lire (L)
              aire←l*L
    cas 'E' : Ecrire ("Entrer la diagonale 1")
              Lire (l)
```

```
            Ecrire ("Entrer la diagonale 2")
            Lire (L)
               aire←PI*l*L div 4
   cas 'C' : Ecrire ("Entrer une côté")
            Lire (c)
            aire←c^2
   Sinon : aire←0
 FinSelon
   Si(aire=0) Alors
    Ecrire ("Forme inconnue")
     Sinon
    Ecrire ("L'aire de la forme géométrique= Réel", aire)
   Finsi
Fin
```

Chapitre 4

Les structures répétitives

4.1) Les itérations 103
4.1.1) Les itérations indéterministes 104
4.1.2) Les itérations déterministes (pour) 110
4.2) Les boucles imbriquées 115
4.3) Choix de la structure itérative 116

CHAPITRE 4

□□□

Les structures répétitives

Résumé : **L'utilisation de plusieurs variables séparées dans le cas d'un problème nécessitant l'inclusion de plusieurs données d'entrées ou de sorties, peut encombrer la mémoire dans certaines situations et peut devenir une tache fastidieuse pour la plupart des utilisateurs. L'utilisation des boucles peut résoudre ce problème et en plus permet de diminuer le nombre d'instructions dans des traitements incluant des actions répétitives. Dans ce chapitre, nous allons présenter, les différents types de boucles, leurs situations d'utilisation ainsi, que leurs syntaxes de définition.**

4.1) Les itérations

Pour exécuter un algorithme plusieurs fois de suite, nous n'avions que la possibilité de le relancer ou de copier n fois certain traitement. Dans le cas des applications nécessitant plusieurs répétitions d'une instruction ou d'un bloc d'instructions, ce procédé peut devenir fastidieux. Il existe un outil algorithmique qui permet de demander à l'ordinateur de ré-exécuter un

certain traitement (soit un nombre de fois déterminé, soit tant qu'une condition est vérifiée), au lieu de copier n fois certain traitement. Toute répétition de l'exécution d'un traitement (une action ou une séquence d'actions) est appelée itération ou boucle. Une itération permet d'ordonner la ré-exécution d'un calcul, grâce à des structures de contrôle. Nous pouvons distinguer deux grandes catégories d'itérations :

- ✓ **Les itérations déterministes** : le nombre de répétitions est défini à l'entrée de la boucle ;
- ✓ **les itérations indéterministes** : l'exécution de la prochaine boucle est conditionnée par une expression booléenne

4.1.1) Les itérations indéterministes

Il existe deux instructions permettant de faire des boucles indéterministes : L'instruction **tant que** et l'instruction **répéter...jusqu'à**.

a) La structure tant que

Cette structure permet de répéter un traitement un certain nombre de fois, tant qu'une certaine condition **cond** est satisfaite. La syntaxe de cette structure se définit comme suit :

```
Tantque (cond=vraie)
Faire
traitement ()
Fintantque
```

Syntaxe de la structure tant que

Un traitement est exécuté tant que la condition cond est évaluée à Vrai. Dès que la condition "cond" devient fausse, la répétition du traitement se termine, et par conséquent l'exécution de la structure "Tantque" s'arrête. La condition "cond" est évaluée avant d'exécuter le traitement. Si cette condition est fausse la première fois, le traitement n'est jamais exécuté. La condition "cond" doit devenir fausse au bout d'un nombre fini d'itérations, sinon l'exécution de cette structure répétitive est infinie ("boucle infinie"). Par conséquent, l'exécution du traitement doit provoquer une modification des données intervenant dans l'expression de la condition cond. Le schéma d'évolution de l'instruction tant que est représenté par la Figure suivante :

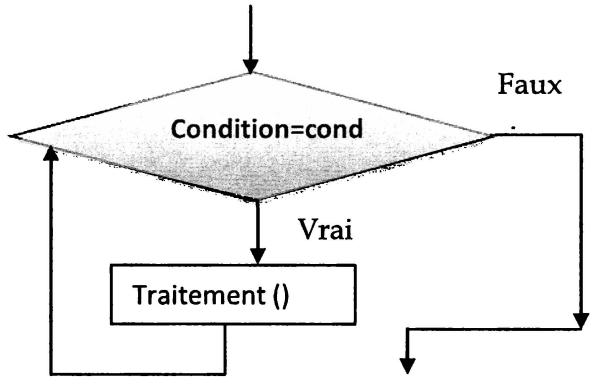


Figure 7 Schéma d'évolution de la structure tant que

- **Exemple du monde réel :**

TantQue NON (Le polycopié d'informatique est assimilé)
Faire
J'étudie le polycopié d'informatique pour l'assimiler
FinTantQue

Règles

- Si vous ne voulez pas que votre algorithme "tourne" indéfiniment, l'expression booléenne doit faire intervenir des variables dont le contenu doit être modifié par au moins une des instructions du corps de la boucle

Application 1 : *Écrire un algorithme qui demande à un utilisateur de répéter un certain nombre de fois, la saisie d'un nombre entier et de faire la somme des nombres saisis autant de fois que la somme est < à une limite fixée d'avance.*

Solution :

D.E : Entier n, limit
D.S : Entier somf
Début
Ecrire ("Saisir une valeur entière limite :")
Lire (limit)
somf←0
TantQue (somf ≤limit)
Faire
Écrire ("Saisir un entier quelconque :")
Lire (n)
somf←somf+n
FinTantQue
Ecrire ("La somme bornée par un entier est un entier", limit, somf-n)
Fin

Application 2 : *Ecrire un algorithme qui permet d'identifier et d'afficher le plus grand entier e tel que e! ≤ n.*

Solution :

D.E: Naturel n
D.S : Naturel e

```
D.I : EntLong fact
Début
Ecrire ("Saisir un entier naturel n≥1:")
Lire(n)
fact←1
e←1
TantQue (fact ≤ n)
  Faire
  e←e+1
  fact←fact*e
 FinTantQue
 e←e-1
  Ecrire ("Le plus grand entier ≤ à un Naturel est un Naturel :", n, e)
Fin
```

	n	e	fact	fact≤n
fact←1	10	?	1	Vrai
e←1	10	1	1	Vrai
e←e+1 fact←fact*e	10	2	2	Vrai
e←e+1 ;fact←fact*e	10	3	6	Vrai
e←e+1,fact←fact*e	10	4	24	Faux
e←e-1	10	3	24	Faux

Tableau 10. Test à la main de l'algorithme plusGrandEntier (cas n=10)

Pour n=10, le résultat de l'exécution de l'algorithme plus haut est représenté par la valeur entière 3 (dernière valeur de e) qui désigne le plus grand entier ayant comme valeur factorielle ≤10. La boucle "Tantque" est exécutée 3 fois et par la suite, l'algorithme exécute 3 itérations.

b) La structure Répéter...jusqu'à

À la différence de l'instruction *tant que*, cette instruction s'adresse aux cas où au moins un traitement doit être effectué. Cette boucle répète l'instruction qu'elle contient (un bloc) tant qu'une condition mentionnée soit fausse. Dans ce schéma itératif, on ne sait pas a priori combien de fois une telle boucle sera répétée. L'instruction de cette boucle est toujours parcourue au moins une fois. En effet, le test de sortie de cette boucle n'est examiné qu'à la fin de chaque répétition. La syntaxe de cette structure se définit comme suit :

Syntaxe de la structure répéter

```
Répéter
Faire
traitement (s)
Jusqu'à (cond=vraie)
```

Règles

- La sortie de la boucle ne se fait qu'après un parcours complet du traitement et non dès que la condition mentionnée devient Vraie.
- Après chaque exécution, on évalue l'expression de la condition et on répète la boucle si l'évaluation de l'expression a comme valeur vraie. En cas où l'évaluation de l'expression donne la valeur fausse, la boucle s'arrête.

Le schéma d'évolution de l'instruction répéter...jusqu'à est représenté par la figure suivante :

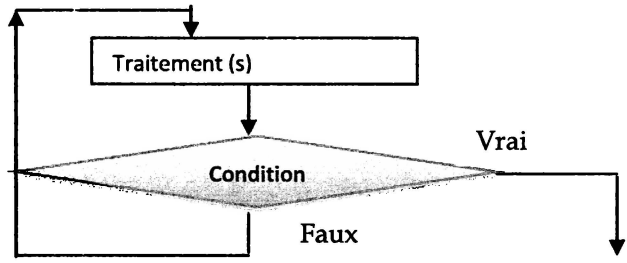


Figure 8. Schéma d'évolution de l'instruction répéter...jusqu'à

- **Application 1** : Écrire *un algorithme qui demande à un utilisateur de répéter la saisie d'un entier jusqu'à l'obtention d'un entier positif.*

Solution :

```
D.E : Entier n
Début
Répéter
  Ecrire ("Saisir un entier :")
  Lire (n)
Jusqu'à (n>0)
Ecrire ("bravo")
Fin
```

- **Application 2** : Écrire *un algorithme qui calcule et affiche le pgcd de deux entiers en utilisant l'algorithme d'Euclide.*

Solution :

```
D.E : Naturel a, b
D.S : Naturel pgcd
D.I : Naturel reste
Début
 Répéter
  Ecrire ("Saisir deux entiers positifs : a et b")
  Lire (a,b)
  Jusqu'à (a>0 et b>0)
  Répéter
```

```
  reste←a%b
  a←b
  b←reste
  Jusqu'à (reste=0)
  pgcd←a
  Ecrire ("le pgcd de Entier et Entier est =Entier", a, b,
pgcd)
Fin
```

	a	b	reste	pgcd	a>0	b>0	reste=0
	10	5	?	?	Vrai	Vrai	Vrai
reste←a%b a←b b←reste	5	0	0	5	Vrai	Vrai	Vrai

Tableau 11. Exécution à la main de l'algorithme pgcd euclidien

Après l'exécution de l'algorithme de calcule de PGCD, le résultat obtenu est représenté par la valeur 5 (dernière valeur de pgcd) qui désigne le pgcd des entiers 10 et 5. La boucle répéter...jusqu'à est exécutée 1 fois, donc l'algorithme exécute une seule itération.

Note :

- les conditions des boucles répéter...jusqu'à et tantque sont contradictoires.

4.1.2) Les itérations déterministes (pour)

Il existe une seule instruction permettant de faire des boucles déterministes, c'est l'instruction "**pour**". La structure de boucle

"pour" est dérivée de la structure de répétition "tant que". Elle permet de répéter une action un nombre fini de fois, ce nombre étant connu à l'entrée de la boucle. Le contrôle de la structure pour est assuré par une variable de type compteur. Par exemple, dans un problème qui demande de calculer la moyenne de N notes relatives à N étudiants, la Saisie de N notes et l'écriture des instructions comme Ecrire (" donner note n° i"), Lire (N) est un travail fastidieux. Pour éviter ce problème, il est performant de répéter N fois l'action d'affichage et de lecture d'une note en adoptant la structure pour. La syntaxe de la boucle "pour" est la suivante :

Syntaxe de la structure pour

```
Pour comp de vali à valf
[Par pas de dec|inc]
Faire
traitement (s)
FinPour
```

Dans cette syntaxe, le traitement est répété exactement n fois, où n désigne une variable entière. La variable "comp" est appelée variable de contrôle de boucle (compteur de boucle). La variable compteuse est, généralement, initialisée (affectée) à 1, et augmentée d'une unité à l'issue de chaque itération. Si n est strictement inférieur à 1, le traitement n'est exécuté aucune fois. Si n vaut 1, le traitement est exécuté une fois et une seule.

D'une manière générale, la variable de contrôle d'une boucle "pour" doit être d'un type discret muni d'une relation d'ordre (Entier, Booléen, Caractère, ...), et les deux bornes "vali" et "valf" déterminant l'intervalle de variation de "comp" sont deux valeurs quelconques de même type. Le schéma décrivant

l'évolution de l'instruction pour est présenté par la figure suivante :

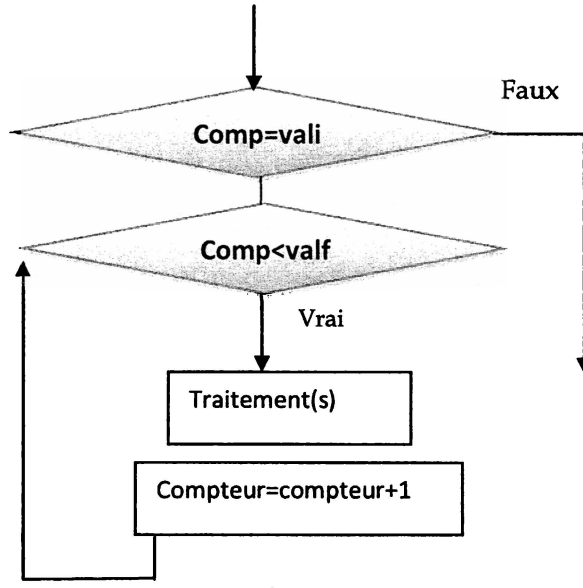


Figure 9. Diagramme d'évolution de l'instruction Pour

Règles

- Dans la structure "pour" décrite précédemment, l'ordre implicite est l'ordre croissant qui consiste à faire progresser le compteur de boucle d'une valeur à la valeur suivante, dans l'intervalle des valeurs considérées, dans ce cas on doit ajouter l'expression optionnelle "**par pas de inc**" (où "inc" désigne l'incrément). Il existe une structure "pour" analogue qui opère dans l'ordre décroissant : à l'issue de chaque itération, la valeur du compteur de boucle est positionnée à la valeur précédente, dans l'intervalle des valeurs considérées. Cette structure "pour" décroissante doit inclure l'expression optionnelle "**par pas de dec**" (où dec désigne le décrément).
- L'exécution commence par l'expression **comp=vali**, qui est là pour permettre l'initialisation de la boucle. L'exécution continue de manière itérative.

- Avant chaque itération, l'expression de la condition "**comp<=valf'**", qui est la condition de continuation de la boucle, est évaluée. Si son résultat est faux, on sort de la boucle, c.-à-d. qu'on passe à l'instruction suivante l'instruction "pour" ; sinon, deux actions successives sont réalisées (exécution des instructions de la boucle et évaluation de l'expression "**comp<valf'**").

Remarque : Pour un pas positif, la valeur négative doit être inférieure à la valeur finale. Pour un pas négatif, la valeur négative doit être supérieure à la valeur finale. Si la valeur initiale est égale à la valeur finale, la boucle sera exécutée une seule fois.

Application 1 : *Ecrire un algorithme qui permet de calculer et d'afficher la somme des n premiers entiers positifs (s=0+1+2+...+n).*

Solution :

```
D.E: Naturel n
D.S: Naturel s
D.I : Naturel i
 Début
  Ecrire ("Saisir l'entier naturel n :")
  Lire (n)
  s←0
  Pour i de 1 à n
   Faire
   s←s+i
  FinPour
  Ecrire ("La somme des n premiers entiers naturels est un
entier :", n, s)
Fin
```

	n	s	i	i≤n
s←0	10	0	1	Vrai
s←s+i	10	1	2	Vrai
s←s+i	10	3	3	Vrai
s←s+i	10	6	4	Vrai
S←s+i	10	10	5	Vrai
s←s+i	10	15	6	Vrai
s←s+i	10	21	7	Vrai
s←s+i	10	28	8	Vrai
s←s+i	10	36	9	Vrai
s←s+i	10	45	10	Vrai
s←s+i	10	55	11	Faux

Tableau 12. Exécution à la main de l'algorithme SomnPremierEntier

Le résultat de l'exécution de l'algorithme de la suite des n premiers entiers retourne la valeur 55 (dernière valeur de s) représentant la somme des 10 premiers entiers naturels. L'instruction "s←s+ i" contenue dans la boucle "pour" est exécutée 10 fois, donc l'algorithme exécute 10 itérations.

- **Application 2** : Écrire un algorithme qui demande successivement 20 nombres à l'utilisateur, puis affiche le plus grand parmi ces 20 nombres :

Entrez le nombre numéro 1 : 12
Entrez le nombre numéro 2 : 14
...
Entrez le nombre numéro 20 : 6
Le plus grand de ces nombres est : 14
En plus, cet algorithme affiche la position indiquant l'ordre de saisie de ce nombre (C'était le nombre numéro 2).

Solution :

```
D.E : Entier a
D.S : Entier max, pmax
DEBUT
  Ecrire ("Donner le nombre numéro 1")
  Lire (a)
 Max← a
 pmax ←1
 Pour i de 2 à 20
  Faire
 Ecrire  (" Entrez le nombre numéro Entier :", i)
 Lire (a)
 SI (a > max) Alors
  max ← a
  pmax ← i
 FinSi
FinPour
Ecrire ("Le plus grand nombre est =Entier :", max)
Ecrire ("Sa position est =Entier :", pmax)
Fin
```

4.2) Les boucles imbriquées

Une boucle peut être incluse dans une autre boucle. Cette inclusion est appelée **imbrication des boucles.** Un exemple typique

de boucles imbriquées consiste à programmer une boucle principale (celle qui intègre les employés un par un) et à l'intérieur, une boucle secondaire (celle qui prend les commandes d'un employé une par une).

- **Exemple :** calcul de la l'expression suivante :
  $\sum_{i=1}^{n} \prod_{j=i}^{n} i + j$

```
Som←0
 Pour i de 1 à n
  Faire
  Prod←1
  Pour j de i à n
   Faire
   Prod←Prod*(i+j)
  FinPour
 Som←Som+Prod
FinPour
```

4.3) Choix de la structure itérative

Les paragraphes plus hauts décrivent trois façons différentes pour programmer des boucles (**Tantque**, **Répéter...jusqu'à** et **Pour**). La structure qui reflète le mieux l'idée du programme à réaliser doit respecter les directives suivantes :

- Utiliser "**Tantque**" ou "**Pour**" : Si le bloc d'instruction (traitement(s)) ne doit pas être exécuté si la condition est fausse.
- Utiliser "**répéter...jusqu'à**" : Si le bloc d'instruction (traitement(s)) doit être exécuté au moins une fois.
- Utiliser **Pour** : Si le nombre d'exécutions du bloc d'instruction (traitement(s)) dépend d'une ou de plusieurs variables qui sont modifiées à la fin de chaque répétition. La boucle "Pour" nous permet de réunir avan-

tageusement les instructions qui influencent le nombre de répétitions au début de la structure.

- Utiliser "**Tantque**" : Si le bloc d'instructions doit être exécuté aussi longtemps qu'une condition extérieure est vraie. La boucle "Tantque" a l'avantage de correspondre plus exactement aux structures de plusieurs langages de programmation.

Le choix entre **pour** et **Tantque** n'est souvent qu'une question de préférence ou d'habitudes.

Exercices du chapitre 4

Exercice 4.1

Ecrire un algorithme qui demande à l'utilisateur de saisir n notes, puis d'afficher la moyenne de ces notes.

Exercice 4.2

Écrire un algorithme qui calcule pour un entier n positif, saisi par l'utilisateur, la valeur factorielle (n!).

Exercice 4.3

Écrire un algorithme qui lit un réel x et un entier positif n et affiche x puissance n.

Exercice 4.4

Écrire un algorithme qui demande la saisie d'un nombre compris entre 1 et 3 jusqu'à ce que la réponse convienne.

Exercice 4.5

Écrire un algorithme qui demande la saisie de deux entiers (notés x et y) et effectue la multiplication positifs entre eux en utilisant l'addition entière.

Exercice 4.6

Écrire un algorithme qui demande un nombre compris entre 15 et 25, jusqu'à ce que la réponse convienne. En cas d'une réponse supérieure à 25, on fera apparaître un message : «Plus petit!», et inversement, «Plus grand!» si le nombre est inférieur à 15.

Exercice 4.7

Écrire un algorithme qui effectue, en même temps, les opérations division entières et modulo à l'aide des soustractions successives à partir d'un premier nombre entier positif et d'un deuxième (le diviseur) entier strictement positif à saisir par un utilisateur. En résultat, l'algorithme doit retourner le résultat du reste de division ainsi que le quotient.

Exercice 4.8

Écrire un algorithme qui demande de lire la suite des prix (en Drs et dont la saisie est terminée par zéro) des achats d'un client. Cet algorithme permet de calculer et d'afficher le reste à rendre, sachant que le client a déjà payé un montant (ce montant doit être lu au clavier).

Exercice 4.9

Écrire un algorithme qui permet de calculer une valeur approchée de pi par la série : $pi^2 = 8\ (1 + (1/3)^2 + (1/5)^2 + (1/7)^2 + (1/9)^2 + \ldots + (1/n)^2)$.

Exercice 4.10

1) Ecrire un algorithme qui demande la saisie d'un nombre binaire, puis de faire la conversion vers un nombre décimal (vérifier la saisie du nombre binaire).

2) Généraliser la conversion pour un nombre décimal d et une base b comprise entre 2 et 10.

Exercice 4.11

Écrire un algorithme selon l'énoncé suivant :

Ayant d'abord introduire le numéro d'une carte bancaire, saisir un certain nombre d'essais ne dépassant pas 3 pour retrouver ce numéro. Si l'utilisateur introduit un numéro correct dans les 3 premiers essais, l'algorithme affiche "ok, bienvenu". Dans le cas contraire et si après le troisième essai on n'a toujours pas trouvé le numéro, l'algorithme affiche "échec".

Exercice 4.12

1) Écrire un algorithme qui demande de saisir un entier n positif et détermine si ce nombre est premier ou non.
2) Généraliser l'algorithme qui reçoit en entrée un entier strictement positif donné, et vérifie si les nombres inférieurs sont premiers ou non.

Exemple : Les nombres premiers inférieurs à 100 : 1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97.

Indication : Un nombre n est premier s'il n'est pas divisible par ses prédécesseurs supérieurs à un. Il suffit donc de regarder si le nombre est divisible par ses prédécesseurs compris entre 2 et n/2.

Exercice 4.13

Écrire un algorithme qui demande à un utilisateur de saisir un entier positif et retourne la partie entière inférieure de la racine carrée de cet entier.

Exemples :

- La partie entière inférieure de la racine carrée de 10 est 3
- La partie entière inférieure de la racine carrée de 16 est 4
- La partie entière inférieure de la racine carrée de 17 est 4

Exercice 4.14

Soit A et B deux nombres positifs, non nuls tel que A>B. Écrire un algorithme qui permet de calculer le plus grand commun diviseur (PGCD).

Indication : Le PGCD de A et B est le nombre qui divise A et B et en même temps. Il est supérieur à tous les autres diviseurs de A et B.

Exercice 4.15

Écrire un algorithme qui demande la saisie d'un entier n non nul et calcule les sommes suivantes :

1) $A=\sum_{i=1}^{n}\prod_{j=0}^{n}(i+j)$

2) $B=\sum_{i=1}^{n}\prod_{j=i+1}^{n}(i+j)$

Exercice 4.16

1) Écrire un algorithme qui demande la saisie d'un entier n>4 et vérifie si ce nombre est parfait ou non.

Indication : On dit qu'un nombre est parfait si la somme de ses diviseurs est égale au nombre lui-même.

Exemple: 6=1+2+3 est parfait

2) Généraliser l'algorithme élaboré dans 1) pour vérifier et afficher tous les nombres parfaits allant de 4 à n.

Exercice 4.17

Écrire un algorithme qui affiche les 10 premières valeurs d'une table de multiplication pour un entier n strictement positif saisi par l'utilisateur.

Exemple : pour n=3 l'algorithme affiche la table de multiplication suivante :

3	6	9	12	15
18	21	24	27	30

Exercice 4.18

Écrire un algorithme qui affiche à l'écran le pattern suivant.
Indication : Utiliser des boucles imbriquées

654321
54321
4321
321
21
1

Exercice 4.19

Écrire un algorithme qui demande de saisir un nombre entier, puis détermine et affiche combien de fois il est divisible par 2.
Exemple :

- 11 est divisible 0 fois par 2
- est divisible 2 fois par 2
- 8 est divisible 3 fois par 2.

Exercice 4.20

Écrire un algorithme qui calcule la valeur u et le rang k du premier terme de la suite de Fibonacci limitée par une borne entière positive b>2 saisie par l'utilisateur.

Rappel : La suite de Fibonacci est définie par : u0 = 1 ; u1 = 1 et $u_n = u_{n-1} + u_{n-2}$ pour n >= 2.

Exercice 4.21

Écrire un algorithme qui demande la saisie d'un entier positif puis le transforme en un entier représentant son complément.
Exemple : Nous appellerons complément de 234, l'entier 766!

Exercice 4.22

Écrire un algorithme qui demande la saisie d'un entier positif et vérifie si ce nombre est palindrome ou non.
Remarque : Un nombre palindrome est un nombre qui reste invariant lorsqu'on l'écrit à l'envers. Par exemple : 1001, 9999, 2002,...

Indication : La taille du nombre saisi n'a pas d'intérêt. La taille maximale du nombre saisi c'est la taille supportée par la capacité d'un entier.

Exercice 4.23

Écrire un algorithme qui permet de calculer pour une valeur réelle donnée (X), la valeur numérique d'un polynôme de degré n :

$P(X) = A_nX^n + A_{n-1}X^{n-1} + ... + A_1X + A_0$

Les valeurs de n, des coefficients A_n,..., A_0 et de X seront entrées au clavier. Utilisez le *schéma de Horner* qui évite les opérations d'exponentiation lors du calcul:

$((((((((A_n*X+A_{n-1})* X)+A_{n-2})*X)+....*X)+A1+)A_0)$

Exercice 4.24

Écrire un algorithme qui permet de calculer la racine carrée X d'un nombre réel positif A par approximations successives en utilisant la relation de récurrence suivante :

$$X_{J+1} = (X_J + \frac{A}{X_J}) / 2 \qquad \text{tel que } X0 = A$$

La précision du calcul J est à entrer par l'utilisateur.

Indications :

Assurez-vous lors de l'introduction des données que la valeur de A est un réel positif et que J est un entier naturel positif, plus petit que 50.

c) Affichez lors du calcul toutes les approximations calculées :

La 1ère approximation de la racine carrée de ... est ...

La 2e approximation de la racine carrée de ... est ...

La 3e approximation de la racine carrée de ... est ...

...

Exercice 4.25

Étant donné que la somme suivante : $1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\cdots+-1^{n}\frac{1}{2n+1}$ tend vers $\frac{\pi}{4}$.

Écrire un algorithme permettant de calculer et d'afficher la valeur de Π : Le calcul s'arrête lorsque $|\frac{-1^{n}}{2n+1}|$ ≤**epsilon** (epsilon est une valeur réelle infinitésimale positive <0.33 saisie au clavier).

Exercice 4.26

Ecrire un algorithme qui demande à un utilisateur de saisir un nombre entier A, puis calcule la racine cubique de ce nombre. La racine cubique d'un nombre peut être calculée en utilisant la suite convergente définie par :

$U_0=1$

$$U_{n+1}=\frac{1}{3}*\left[2U_n+\frac{A}{Un^2}\right]$$

Cette suite converge vers $\sqrt[3]{A}$.

Le calcul s'arrête lorsque $|A-(U_{n+1})^3|$ ≤epsilon. Où epsilon est un nombre réel infinitésimal saisi au clavier (epsilon<0.2).

Exercice 4.27

Écrire un algorithme qui permet de vérifier et d'afficher tous les entiers k compris entre 1 et un entier n (dont la valeur sera saisie au clavier), tels que la somme des cubes des chiffres composant k est égale à k lui-même. Ex : $153 = 1^3 + 5^3 + 3^3$.

Exercice 4.28

Un entier naturel est divisible par 11 si la somme de ses chiffres de rang pair est égale à celle de ses chiffres de rang impair, modulo 11. Écrire un programme qui saisit un nombre entier naturel a et indique s'il est divisible par 11 ou non.

Corrigé des exercices du Chapitre 4

Exercice 4.1 (Solution)

D.E : **Entier** n
D.I : **Entier** i
D.S : **Entier** res
Début
  Ecrire ("Entrer un entier n")
  Lire (n)
  res←0
  Pour i de 1 à n
  Faire
    res←res+i
  FinPour
Fin

➡ *Exécution à la main :* somme des 4 entiers naturels =10

n	res	i
4	0	1
	1	2
	3	3
	6	4
	10	

Exercice 4.2 (Solution)

Résolution : n! = n * (n-1) *....*1

Démarche : deux solutions :

- Utiliser une boucle "pour" dont le compteur commence de 1 à n
- Ou utiliser une boucle "pour" dont le compteur commence de n à 1 :

D.E : **Naturel** n

D.I : **Entier** i

D.S : **NaturelLong** fact

Début

Ecrire ("Entrer un entier n")
Lire (n)
fact←1
Pour i de 1 à n
Faire
fact←fact*i
FinPour

Fin

- ***Exécution à la main :*** La factorielle de 4 = 24

n	fact	i
4	1	1
	1	2
	2	3
	6	4
	24	

Exercice 4.3 (Solution)

Résolution : $x^n = x*x*....*x$ } y fois

Démarche : utiliser une boucle "pour" dont le compteur commence de 1 à n et qui multiplie, à chaque itération, la dernière puissance obtenue (puissance d'ordre i) par x :

```
D.E/D.S : Réel x
          Entier n, puiss
D.I : Entier i
Début
  Ecrire ("Entrer un entier n")
  Lire (n)
  Ecrire ("Entrer un réel x")
  Lire (x)
   puiss←1
 Pour i de 1 à n
  Faire
    puiss←puiss*x
  FinPour
 Ecrire ("Le résultat d'un Réel puissance un Entier est
=Réel",x,n,x)
 Fin
```

Exercice 4.4 (Solution)

```
D.E : Réel a
 Début
Tant Que (a < 1 OU a > 3)
  Ecrire ("Veuillez Saisir une valeur comprise entre 1 et 3 ")
```

Lire (a)
FinTantQue
Fin

Exercice 4.5 (Solution)

Résolution : x*y = x*x*....*x} y fois

Démarche : utiliser une boucle "pour" dont le compteur commence de 1 à y et qui incrémente le compteur "mult" (initialisé à 0 ; représentant la somme d'ordre i) du produit final, à chaque itération, par x :

D.E : Entier x, y
D.S : Entier mult
Début
Ecrire ("Entrer un entier x")
Lire (x)
Ecrire ("Entrer un entier y")
Lire (y)
mult←0
Pour i de 1 à y **Faire**
mult←mult+x
FinPour
Fin

➡ ***Exécution à la main :*** La multiplication de 3 par 4 =**12**

x	y	i	mult
3	4	1	0

		2	3
		3	6
		4	9
			12

Exercice 4.6 (Solution)

```
    D.E : Réel a
Début
    Ecrire ("Donner un réel a")
    Lire (a)
    TantQue (a < 15 OU a > 25)
     Faire
     Si (a < 15) Alors
       Ecrire ("Plus petit")
     Finsi
      Si (a>25) Alors
     Ecrire ("Plus grand")
     FinSi
     Ecrire ("Donner un réel a")
     Lire (a)
     FinTantQue
    Fin
```

Exercice 4.7 (Solution)

D.E : Naturel n

```
D.S : Naturel divis, reste, quotient
Début
  Répéter
    Ecrire ("Donner un entier positif")
    Lire (n)
    Ecrire ("Donner un entier strictement positif")
    Lire (divis)
   Jusqu'à (divis>0 ET n >=0)
  reste←n
  quotient←0
 TantQue (reste≥divis)
   Faire
    reste←reste-divis
    quotient←quotient+1
  FinTantQue
 Ecrire ("Entier divisé par Entier= Entier et reste Entier", n,
divis,  quotient, reste)
Fin
```

Exercice 4.8 (Solution)

```
D.E : Entier prix, mpaye
D.I : Entier mach
D.S : Entier reste
Début
 mach←0
 Repeter
  Ecrire ("Entrez le prix")
  Lire (prix)
  mach ←mach + prix
 Jusqu'à (prix=0)
 Ecrire ("Entrez le prix payé")
```

```
  Lire (mpaye)
  reste ← mpaye - mach
Ecrire ("Le reste rendu est =Entier", reste)
Fin
```

Exercice 4.9 (Solution)

```
D.E : Entier n
D.I : Entier i
      Réel pi2
D.S : Réel pi
Début
 mach←0
 Repeter
 Faire
  Ecrire ("Entrez un entier n")
  Lire (n)
 Jusqu'à (n>0)
pi2←0
Pour i de 1 à n par pas de 2
 Faire
 pi2←pi2+(1 div i)^2
 FinPour
pi←pi2^0.5
Ecrire ("La valeur approchée de pi est =Réel : ", pi)
Fin
```

Exercice 4.10 (Solution)

1)

```
D.E : Entier n, nb
      Entier d
```

```
D.I : Entier rang
      Booléen bin
D.S : Entier nd
Début
 Ecrire ("Entrez un nombre binaire n")
 Lire (n)
 nb←n     //stocke l'entier original
 bin←Vrai
 Repeter
 Faire
 d←n%10
 Si (d<>0 et d<>1) Alors
  bin←Faux
  n←n/10
 Finsi
 Jusqu'à (bin=Faux ou n=0)
 Si (bin==Vrai) Alors
  Débutalors
   nd←0
 rang←0
 Répéter
  d←n/10
  nd←nd+d*(2^rang)
  nb←nb/10
  rang←rang+1
 Jusqu'à (nb=0)
 Ecrire ("Entier en binaire    =    Entier en décimal ", nb,
nd)
 Finalors
 Sinon
```

Vérifie si un nombre saisi est binaire ou non

Ecrire ("Erreur de saisie du nombre binaire : un nombre binaire doit contenir uniquement des 0 et 1")

Finsi

Fin

➡ ***Exécution à la main*** : 1011 en binaire admet 11 en décimal

n	nb	rang	d	nd
1011	1011	0	1	$0+1*2^0=1$
	101	1	1	$1+1*2^1=3$
	10	2	0	$3+0*2^0=3$
	1	3	1	$3+1*2^3=11$
	0			**11**

2)

D.E : **Entier** b, nb
Entier d
D.I : **Entier** rang
D.S : **Entier** nd
Début
nd←0
rang←0
Répéter
Ecrire ("Entrez la base")
Lire (b)

```
Jusqu'à (b>2 et b<10)
Ecrire ("Entrez un nombre n dans la base Entier", b)
Lire (nb)
Répéter
 d←n/10
 nd←nd+d*(b^rang)
 nb←nb/10
 rang←rang+1
Jusqu'à (nb=0)
 Ecrire ("Entier en base Entier admet en décimal Entier",
nb, b, nd)
Fin
```

Exercice 4.11 (Solution)

```
D.E : Entier codbalus  //code bancaire lus
      Constante Entier codBanc=5723 //code bancaire
//enregistré dans la base de données de la b
D.I : Entier cptessais
Début
cptessais←0
 Répéter
 Ecrire ("Entrez votre code bancaire")
 Lire (codbalus )
 cptessais← Cptessais+1
 Jusqu'à (codbalus= codBanc OU cptessais>3)
Si(cptessais≤3) Alors
 Ecrire ("Ok, bienvenue dans notre système ban caire")
 Sinon
```

Ecrire ("Code invalide, veuillez vérifier la validité de votre compte")
FinSi
Fin

Exercice 4.12 (Solution)

1)

D.E : **Entier** n
D.I : **Entier** divpred
Booléen premier
Début
Repeter
Ecrire ("Entrez un entier n")
Lire (n)
Jusqu'à (n>0)
premier←vrai
divpred←2
Si (n=1 **OU** n=2 **OU** n=3) **Alors**
Ecrire ("**Entier** est premier",n)
Sinon
Débutsinon
TantQue(premier=vrai Et divpred<n/2)
Faire
Si (n%divpred=0) **Alors**
premier←Faux
Finsi
divpred←divpred+1
FinTantQue
Si (premier=Vrai) **Alors**
Ecrire ("**Entier** est premier", n)

```
    Sinon
    Ecrire ("Entier est non premier", n)
    Finsi
   Finsinon
  Finsi
 Fin
```

2)

```
D.E : Entier n
D.I: Entier i, divpred
     Booléen premier
Début
 Répéter
  Ecrire ("Entrez un entier n")
  Lire (n)
  Jusqu'à (n>0)
  Pour i de 1 à n-1
 Faire
 premier←vrai
Si (i=1 OU i=2 ou i=3) Alors
Ecrire ("Entier est premier",i)
Sinon
Débutsinon
 divpred←2
  TantQue(premier=vrai Et divpred<=i/2)
 Faire
 Si (n%divpred=0) Alors
  premier←Faux
  Finsi
  divpred←divpred+1
  FinTantQue
  Si (premier=Vrai) Alors
```

```
    Ecrire ("Entier est premier", i)
    Sinon
    Ecrire ("Entier est non premier", i)
    Finsi
   Finsinon
  Finsi
FinPour
 Fin
```

Exercice 4.13 (Solution)

Résolution : On démarre notre résolution en se basant sur l'hypothèse suivant : un entier n est toujours supérieur ou égal à la carrée de sa racine carrée : ($\sqrt{n} * \sqrt{n} \leq n$)

```
D.E : Entier n
D.S: Entier racineinf
Début
  Repeter
   Ecrire ("Entrez un entier n")
   Lire (n)
   Jusqu'à (n>0)
   racineinf←1
  TantQue(n≥racineinf*racineinf)
   Faire
   racineinf←racineinf+1
   FinTantQue
    Ecrire ("La racine carrée inférieure de Entier", racineinf-
1)
 Fin
```

➡ ***Exécution à la main :*** La racine carrée inférieure de 10 est 3

n	racineinf	n>racineinf*racineinf
10	1	10≥1 ➔vrai
	2	10≥4➔vrai
	3	10≥9➔Vrai
	4	10≥16➔Faux
racineinf-1	3	

Exercice 4.14 (Solution)

```
D.E : Entier A, B
D.S: Entier reste
Début
  Repeter
   Ecrire ("Entrez deux entiers")
    Lire (A, B)
   Jusqu'à (A>B et B>0)
   reste←A-B
  TantQue(reste>B)
   Faire
    A←reste
    reste←A-B
   FinTantQue
  Ecrire ("Le PGCD de Entier et Entier est= Entier", A, B,
reste)
 Fin
```

➡ ***Exécution à la main :*** - Le PGCD de 15 et 10 est= 5
- Le PGCD de 9 et 6 est =3
- Le PGCD de 6 et 2 est= 2

A	B	reste	reste>B
15	10	5	5>10 ➔Faux
9	6	3	3>6→Faux
6	2	4	4>2→Vrai
4		2	2>2→Faux

Exercice 4.15 (Solution)

1)

D.E : **Entier** n
D.I: **Entier** i, j
D.S : **Entier** som, prod
Début
Repeter
Ecrire ("Entrez un entier n")
Lire (n)
Jusqu'à (n>0)
som←0
Pour i de 1 à n
Faire
prod←1
Pour j de 0 à n
Faire
prod←prod*(i+j)
FinPour

```
    som←som+prod
   FinPour
    Ecrire ("La somme des produits d'ordre Entier est
=Entier", n, som)
  Fin
 2)

  D.E : Entier n
  D.I : Entier i, j
  D.S : Entier som, prod
  Début
    Repeter
     Ecrire ("Entrez un entier n")
     Lire (n)
    Jusqu'à (n>0)
    som←0
   Pour i de 1 à n
    Faire
     prod←1
     Pour j de i+1 à n
      Faire
       prod←prod*(i+j)
     FinPour
    som←som+prod
   FinPour
    Ecrire ("La somme des produits d'ordre Entier est = En-
tier", n, som)
  Fin
```

Exercice 4.16 (Solution)

1)

```
  D.E : Entier n
```

```
D.I : Entier somdiv, div
Début
  Repeter
   Ecrire ("Entrez un entier n")
   Lire (n)
  Jusqu'à (n>4)
  somdiv←0
  Pour div de 1 à n/2
   Faire
   Si (n%div=0) Alors
   somdiv←somdiv+div
   FinSi
  FinPour
   Si (n=somdiv) Alors
    Ecrire ("Entier est parfait", n)
    Sinon
    Ecrire ("Entier est non parfait", n)
 FinSi
Fin
```

2) Généralisation pour n=4 à n

```
D.E : Entier n
D.I : Entier somdiv, div, i
Début
  Repeter
   Ecrire ("Entrez un entier n")
   Lire (n)
  Jusqu'à (n>4)
  Pour i de 4 à n-1
  Faire
  somdiv←0
  Pour div de 1 à i/2
```

```
  Faire
  Si (i%div=0) Alors
   somdiv←somdiv+div
  FinSi
  FinPour
  Si (i=somdiv) Alors
    Ecrire ("Entier est parfait", i)
    Sinon
    Ecrire ("Entier est non parfait", i)
  FinSi
 FinPour
Fin
```

Exercice 4.17 (Solution)

```
D.E : Entier n
D.I : Entier i
Début
  Repeter
   Ecrire ("Entrer un entier n")
   Lire (n)
   Jusqu'à (n>0)
  Pour i de 1 à 10
  Faire
   Ecrire ("Entier", i*n)
   Si (n=5) Alors
    Ecrire (RC)   //retour chariot ou retour ligne
   FinSi
  FinPour
Fin
```

Exercice 4.18 (Solution)

```
D.I : Entier n
Début
 Pour i de 6 à 1 par pas de -1
  Faire
  Pour j de i à 1 par pas de -1
   Faire
    Ecrire (j)
   FinPour
   Ecrire (RC)   //retour à la ligne suivante
  FinPour
Fin
```

Exercice 4.19 (Solution)

```
D.E : Entier n
D.I : Entier i
D.S : Entier nbfois
Début
 Repeter
  Ecrire ("Entrer un entier n :")
  Lire (n)
  Jusqu'à (n>0)
 nbfois←0
 Tantque (n%2=0)
  Faire
   nbfois←nbfois+1
   n←n/2
 FinTantQue
```

Ecrire ("Le nombre saisi est divisible **Entier** fois par 2", nbfois)

Fin

➥ *Exécution à la main :*

- Entrer un entier n : 8

- L'entier saisi est divisible 3 fois par 2

n	nbfois	n%2=0
8	0	8%2=0➔Vrai
4	1	4%2=0➔Vrai
2	2	2%2=0➔Vrai
1	3	1%2=0➔Faux
nbfois	3	

Exercice 4.20 (Solution)

D.E : Naturel b
D.I : Entier u0, u1
D.S : **Entier** k, u
Début
Répéter
Ecrire ("Entrer une borne b :")
Lire (b)
Jusqu'à (b>2)
u0←1
u1←1

k←1
u←u0+u1
TantQue (u<b)
Faire
u0←u1
u1←u
u←u0+u1
k←k+1
FinTantQue
Ecrire ("La valeur de la suite de Fibonacci bornée par **Entier** est d'ordre **Entier** ayant la valeur **Entier**", b, k-1, u- u0)
Fin

➡ *Exécution à la main :*

- Entrer une borne b : 10

- La valeur de la suite de Fibonacci bornée par 10 est d'ordre 4 ayant la valeur 8.

k	u	u0	u1	b	u<b
1	2	1	1	10	Vrai
2	3	1	2		Vrai
3	5	2	3		Vrai
4	8	3	5		Vrai
5	13	5	8		Faux

Exercice 4.21 (Solution)

Solution 1

D.E : Naturel n, nc
D.I : Naturel mult
D.S : **Naturel** compl
Début
Répéter
Ecrire ("Entrer un entier")
Lire (n)
Jusqu'à (n>0)
nc←n //pour conserver la valeur originale de n
mult←1
compl←0
Répéter
Si (mult=1) **Alors**
compl←compl+mult*(10-n%10)
Sinon
compl←compl+mult*(9-n%10)
n←n/10
mult←mult*10
Finsi
Jusqu'à(n=0)
Ecrire ("Le complément de **Entier** est **Entier**", nc, compl)
Fin

➡ ***Exécution à la main :***

- Entrer un entier : 234
- Le complément de 234 est 766

n	mult	nc	compl	mult=1	n%10<>0
234	1	**234**	0	Vrai	4<>0➔ Vrai
23	10		9-3=6	Faux	3<>0➔Vrai

2	100		6+6*10=66	Faux	2<>0➔Vrai
0			700+66=766		0<>0➔Faux

Solution II

D.E : Naturel n
D.I : Naturel mult, nc, ns
D.S : Naturel compl
Début
Répéter
Ecrire ("Entrer un entier")
Lire (n)
Jusqu'à (n>0)
ns←n //sauvegarde de la valeur de n
nc←0 //compteur sur le nombre des chiffres de n
Répéter
n←n/10
nc←nc+1
Jusqu'à (n=0)
Ecrire ("Le complément d'un **Entier** est un **Entier**", ns, 10^nc-n)
Fin

Exercice 4.22 (Solution)

D.E : Naturel n, nc
D.I : Naturel mult, taille←0
D.S : Naturel ninv

```
Début
  Répéter
    Ecrire ("Entrer un entier")
    Lire (n)
  Jusqu'à (n>0)
  nc←n  //pour conserver la valeur originale de n
  Répéter
   taille←taille+1
   n=n/10
  Jusqu'à (n=0)
  mult←10^(taille-1)
  ninv←0
  n←nc
  TantQue (n <>0)
   Faire
    ninv←ninv+mult*(n%10)
     n←n/10
    mult←mult/10
   FinTantQue
  Si (nc=ninv) Alors
   Ecrire ("Entier est palindrôme", nc)
   Sinon
   Ecrire ("Entier est non palindrôme", nc)
  FinSi
Fin
```

➡ *Exécution à la main :*

- Entrer un entier : 133
- 133 est non palindrome

n	mul t	nc	ninv	n%10<>0	nc=ninv
133	100	133	0	3<>0➔ Vrai	
13	10		3*100=300	3<>0➔Vrai	
1	1		3*10+300= 330	1<>0➔Vrai	
0			300+30+1 =331	0<>0➔Faux	**Faux**

Exercice 4.23 (Solution)

D.E : Réel X, A
Entier n
D.I : Entier i
D.S : **Réel** hornval
Début
Répéter
Ecrire ("Entrer un entier :")
Lire (n)
Jusqu'à (n>0)
Ecrire ("Entrer un réel X :")
Lire (X)
Ecrire ("Entrer le coefficient d'ordre **Entier** : ", n)
Lire (A)
hornval←A
Pour i **de** n-1 **à** 0 **Faire**
Faire

Ecrire ("Entrer le coefficient d'ordre **Entier** : ", i)
Lire (A)
hornval←(hornval*X)+A
FinPour
Ecrire ("Pour X=**Réel** et n=**Entier**, le polynôme de Horner = **Réel**", X, n, hornval)
Fin

➥ ***Exécution à la main :***

- Entrer un entier : 4
- Entrer un réel X : 1.5
- Entrer le coefficient d'ordre 4 : 1.6
- Pour X=1.5 et n=4, le polynôme de Horner = 13.625

n	A	X	i	hornval
4	A4=1.6	1.5	3	1.6
	A3=1.3		2	1.6*1.5+1.3
	A2=1.2		1	(1.6*1.5+1.3)*1.5+1.2
	A1=1.8		0	((1.6*1.5+1.3)*1.5+1.2)*1.5 +1.8
	A0=1.7			(((1.6*1.5+1.3)*1.5+1.2)*1.5 +1.8)+1.7=13.625

Exercice 4.24 (Solution)

D.E : Réel A
Entier J
D.I : Réel X0

```
D.S : Réel X
Début
 Répéter
   Ecrire ("Entrer un réel A:")
   Lire (A)
  Jusqu'à (A>0)
  Répéter
   Ecrire ("Entrer un entier J:")
   Lire (J)
  Jusqu'à (J>0 et J<50)
  X0←A
 X←(X0+A div X0) div 2
 Ecrire ("La 1ère approximation de la racine carrée de Réel
est =Réel", A, X)
 Pour i de 1 à J Faire
   Faire
   X0←X
 X←(X0+A div X0) div 2
 Ecrire ("La Entier éme approximation de la racine carrée de
Réel est = Réel", i+1, A, X)
 FinPour
 Ecrire ("La racine carrée de Réel pour la Entier ème ap-
proximation= Réel", A,J, X)
Fin
```

Exercice 4.25 (Solution)

➡ **Solution 1** : avec boucle tant que

```
D.E : Réel epsilon
D.I : Entier n, rapp
D.S : Réel som
Début
```

Répéter
Ecrire ("Entrer une valeur réelle positive infinitésimale :")
Lire (epsilon)
Jusqu'à (epsilon >0 et epsilon<0.33)
n←0
rapp←(-1)^n div (2*n+1)
som←rapp
Si (rapp<0) Alors
rapp←-rapp
Finsi
TantQue (rapp>epsilon)
Faire
n←n+1
rapp←(-1)^n div (2*n+1)
som←som+rapp
Si (rapp<0) **Alors**
rapp←-rapp
Finsi
FinTantQue
pi←som*4
Ecrire ("La valeur finale de pi approchée est = **Réel**", pi)
Fin

➡ ***Exécution à la main :***

- Entrer une valeur réelle positive infinitésimale : 0.3

epsilon	som	rapp	n	rapp≥epsilon
0.012	1	1	0	1>0.3➔Vrai

	1-0.33=0.67	-0.33	1	0.33>0.3➔Vrai
	0.67+0.2=0.87	0.2	2	0.88>0.3➔Vrai
	0.87-0.14=0.72	-0.14	3	0.72>0.3➔Vrai
	0.72+0.11=0.83	0.11	4	0.83>0.3➔Vrai
	0.72-0.09=0.63	-0.09	5	0.63>0.3➔Vrai
	0.63+0.07=0.7	0.7	6	0.7>0.3➔Vrai
	0.7-0.06=0.64	-0.06	7	0.64>0.3➔Vrai
	0.64+0.067=0.7	0.067	8	0.7>0.3➔Vrai
	0.7-0.05=0.65	-0.05	9	0.65>0.3➔Vrai
	065+0.04=0.69	0.04	10	0.69>0.3➔Vrai
	.....	....	... ...	.....
	.....	....	... ...	rapp<0.3==Vrai

➥ **Solution 2** : avec boucle Répéter…Jusqu'à

D.E : Réel epsilon
D.I : Entier n, rapp
D.S : **Réel** som
Début
Répéter
Ecrire ("Entrer une valeur réelle positive infinitésimale :")

```
    Lire (epsilon)
   Jusqu'à (epsilon >0 et epsilon<0.33)
    n←0
    som←0
   Répéter
    Faire
     rapp←(-1)^n div (2*n+1)
     som←som+rapp
     n←n+1
     Si (rapp<0)  Alors
      rapp←-rapp
     Finsi
     Jusqu'à (rapp≤epsilon)
    pi←som*4
     Ecrire ("La valeur finale de pi approchée est = Réel", pi)
Fin
```

Exercice 4.26 (Solution)

```
D.E : Réel epsilon, Entier A
D.I : Réel u0, u, valcomp //valeur comparée avec epsilon
D.S : Réel racine
Début
  Répéter
   Ecrire ("Entrer une valeur réelle positive infinitésimale :")
   Lire (epsilon)
  Jusqu'à (epsilon >0 et epsilon<0.2)
   Répéter
    Ecrire ("Entrer un entier :")
    Lire (A)
```

```
    Jusqu'à (A>0)
     u0←1
     Répéter
     Faire
     u←(1div 3)*[2*u0+A div u0^2]
     u0←u
     valcomp←A-u^3
     Si (valcomp<0)  Alors
      valcomp←-valcomp
     Finsi
    Jusqu'à (valcomp≤epsilon)
   racine←u^3
  Ecrire ("La valeur de la racine carrée approchée de Entier
est Réel", A, racine)
  Fin
```

Exercice 4.27 (Solution)

```
  D.E : Entier n
  D.I : Entier aux, som
  D.S : Entier i
  Début
    Repeter
     Ecrire ("Entrer un entier n")
     Lire (n)
     Jusqu'à (n>2)
    Pour i de 2 à n-1
    Faire
     aux←i
     som←0
     TantQue (aux%10<>0)
      Faire
```

```
      som←som+(aux%10)^3
      aux←aux/10
     FinTantQue
     Si(i=som) Alors
      Ecrire ("Entier est validé comme = à la somme des cubes de ses chiffres", i)
    Finsi
   FinPour
 Fin
```

Exercice 4.28 (Solution)

```
 D.E : Entier n
 D.I : Entier somcrp, somcri //somme des chiffres de rang //paire et impaire
       Entier rang
       Entier ns //pour sauvegarder la valeur de n
 Début
  Repeter
   Ecrire ("Entrer un entier n")
   Lire (n)
   Jusqu'à (n>10)
   ns←n
   somcrp←0
   somcri←0
   rang←1
   TantQue (n%10<>0)
    Faire
     Si (rang%2==0) Alors
      somcrp← somcrp+n%10
      Sinon
```

```
    somcri← somcri+n%10
   FinSi
   rang←rang+1
  FinTantQue
  Si(somcrp%11=somcri%11) Alors
   Ecrire("Entier est divisible par 11", ns)
  Sinon
   Ecrire("Entier est non divisible par 11", ns)
  Finsi
 FinPour
Fin
```

Chapitre 5

Les sous-algorithmes

5.1) Les sous-algorithmes **164**

5.2) Les sous-algorithmes : types et règles d'utilisation **165**

5.2.1) Règles de nommage des sous-algorithmes 165

5.2.2) Les fonctions **166**

5.2.3) Les procédures **168**

5.2.4) Structure d'un algorithme principal **169**

5.2.5) Définition de la portée des variables **170**

5.2.6) Les paramètres formels/effectifs **172**

CHAPITRE 5

□□□

Les sous-algorithmes (Procédures et fonctions)

"...diviser chacune des difficultés que j'examinerai en autant de parties qu'il se pourrait et qu'il serait requis pour les mieux résoudre."

Descartes

Résumé : L'utilisation des sous-algorithmes dans des problèmes complexes facilite énormément la tâche de résolution. En effet, l'utilisation d'un sous-algorithme (module dans le contexte de la programmation modulaire) permet de décomposer la résolution d'un grand problème à des petites solutions qui résout des sous-problèmes qui font partie du problème principal. Ces sous-algorithmes sont réalisables avec un effort de raisonnement flexible et gérable. L'ensemble des sous-algorithmes produits constitue un algorithme complet qui résout le problème principal. Ce chapitre décrit les types des sous-algorithmes ainsi que les règles relatives à leurs utilisations.

5.1) Les sous-algorithmes

La rédaction d'un algorithme en un seul coup devient difficile à comprendre dès qu'il dépasse un nombre bien déterminé de lignes. Afin de simplifier la résolution d'un problème, l'écriture d'un algorithme qui résout un problème revient toujours à écrire des sous-algorithmes qui résolvent des sous-parties du problème initial. Une écriture modulaire (décomposition en modules) permet de partager un algorithme en plusieurs parties et de regrouper dans l'algorithme principal les instructions en décrivant les enchaînements. Un algorithme est décomposé entre différents sous-algorithmes appelés par l'algorithme principal. Ces sous-algorithmes sont appelés "PROCEDURES" ou "FONCTIONS" permettent d'éviter des séquences d'instructions répétitives. A titre d'exemple, les sous-algorithmes "E*crire*" et "*Lire*" vus dans le chapitre 2 représentent des exemples de sous-algorithmes.

- **Exemple :** *Écrire un algorithme qui affiche les nombres déficients (un nombre déficient est un nombre dont sa valeur est plus grande que la somme de ses diviseurs) compris entre 0 et une valeur n saisie par l'utilisateur.*

La résolution de ce problème revient à résoudre les problèmes suivants :

- *Sous-Algorithme 1* : Demander à l'utilisateur de saisir un entier n
- *Sous-Algorithme 2* : Afficher les nombres déficients compris entre 0 et n
- *Sous-Algorithme 3* : Savoir si un nombre donné est déficient
- *Sous-Algorithme 4* : Calculer la somme des diviseurs d'un nombre

- *Sous-Algorithme 5* : Savoir si un nombre est diviseur d'un autre nombre

Chacun de ces sous-problèmes devient un nouveau problème à résoudre. La résolution de ses sous-problèmes revient à résoudre le problème initial.

5.2) Les sous-algorithmes : types et règles d'utilisation

En algorithmique, il existe deux types de sous-algorithmes : Les fonctions et les procédures. Un sous-algorithme est obligatoirement caractérisé par un nom (un identifiant) unique et dès qu'il sera défini, son nom devient une nouvelle instruction, qui peut être utilisée dans d'autres sous-algorithmes. Le sous-algorithme qui utilise un sous-algorithme. est appelé **sous-algorithme appelant.**

5.2.1) Règles de nommage des sous-algorithmes

D'une manière similaire que les variables, les constantes et les types définis par l'utilisateur, les sous-algorithmes possèdent un nom. Ces noms doivent suivre certaines règles :

- Ils doivent être explicites (à part quelques cas particuliers, comme par exemple les variables *i* et *j* pour les boucles)
- Ils ne peuvent contenir que des lettres et des chiffres
- Ils commencent obligatoirement par une lettre
- Les variables et les sous-algorithmes commencent toujours par une minuscule
- Les types commencent toujours par une majuscule

- Lorsqu'ils sont composés de plusieurs mots, on utilise les majuscules (sauf pour les constantes) pour séparer les mots (par exemple JourDeLaSemaine)

5.2.2) Les fonctions

Conceptuellement, une fonction est une abstraction destinée à calculer une valeur unique à partir d'un ensemble de données fournies en entrées (paramètres). Il s'agit du cas le plus classique des fonctions, qui s'apparentent aux fonctions mathématiques. Soit l'exemple de la fonction à une variable F(x)=2x+3, définie de R dans R. En d'habitude, une telle fonction mathématique est définie de la manière suivante :
F ➔(R➔R : x➔2x+ 3). En algorithmique, une fonction se définit par les caractéristiques suivantes :

- Une fonction possède un seul type, qui est le type de la valeur retournée
- Le passage de paramètre est **uniquement en entrée** : c'est pour cela qu'il n'est pas précisé
- Lors de l'appel, on peut donc utiliser comme paramètre des variables, des constantes, mais aussi des résultats de fonction
- La valeur de retour est spécifiée par l'instruction «**Retourner**»
- Le nom d'une fonction est soit un nom (par exemple *minimum*), soit une question (par exemple *estVide, estPremier...*)

On utilise une fonction en précisant son nom suivi des paramètres entre parenthèses. Les parenthèses sont toujours présentes même lorsqu'il n'y a pas de paramètres. La syntaxe de déclaration d'une fonction se définit comme suit :

Structure d'une fonction algorithmique

```
Fonction Type_de_retour nom_fonction
(paramètre(s)
 de la fonction)
Déclaration type1 variable_locale_1; . . .
Début
  Instructions de la fonction avec l'existen
gatoire
  de l'instruction Retourner
FinFct
```

À titre d'exemple, la fonction F présentée plus haut peut être exprimée en algorithmique comme suit :

```
Fonction Réel f(Réel x)
 Réel r
Début
  r ←2.0*x+3.0
  Retourner (r)
Fin
```

- **Application :** *Ecrire une fonction qui permet de calculer la factorielle d'un entier positif n reçu en paramètre.*

- **Solution :**

```
Fonction EntLong Factorielle (Entier nb)
Variables :
 EntLong x
 Entier i
Début
 x←1
 Pour i de 1 à nb
 Faire
 x←x*i
```

FinPour
Retourner (x)
FinFct

5.2.3) Les procédures

On parle de procédures pour de tels sous-algorithmes qui ne retournent aucun résultat, car on s'éloigne ici de la signification mathématique des fonctions, qui possèdent en général au moins une variable et une valeur. Conceptuellement, une procédure est une abstraction qui représente une action sur des données. Elle peut utiliser les données en entrée seulement, en entrée et en sortie ou en sortie seulement. Généralement, le nom d'une procédure est un verbe. La syntaxe de déclaration d'une procédure se présente comme suit :

```
Procédure nom_de_la_procédure (E paramètre(s)
 En entrée type var, S paramètre(s) en sortie
type var, E/S paramètre(s) en entrée/sortie
type var)
Déclaration variable(s) locale(s)
Début
<Instructions de la procédure>
FinProc
```

Syntaxe de déclaration d'une procédure

Une procédure est appelée de la même manière qu'une fonction, en indiquant son nom suivi des paramètres entre parenthèses.

- **Application 1 :** *Ecrire une procédure qui permet de calculer le minimum et le maximum parmi 3 entiers passés comme paramètres en entrée.*

Solution :

Procédure calculerMinMax3 **(E Entier** a,b,c, **S Entier** min, max**)**
Début
min←minimum3(a,b,c)
max←maximum3(a,b,c)
FinProc

- a, b et c désignent des données d'entrées, alors que min et max désignent des résultats qui devaient être affichés en sortie.

- minimum3 et maximum3 sont deux fonctions déjà définies et qui retournent, respectivement, le minimum et le maximum de 3 entiers en paramètres

Application2 : Ecrire une procédure qui permet de permuter deux entiers passés en paramètre.

Solution :

Procédure echanger **(E/S Entier** a, **E/S Entier** b**)**
Entier vaux
Début
vaux←a
a←b
b←vaux
Fin

5.2.4) Structure d'un algorithme principal

Un algorithme principal doit suivre la structure suivante :

Structure d'un algorithme principal

```
Définition des constantes
Définition des types
Déclaration des variables globales
Définition des sous-algorithmes
Début
<Instructions de l'algorithme principal>
Fin
```

5.2.5) Définition de la portée des variables

- La **portée** d'une variable est l'ensemble des sous-algorithmes où cette variable est connue (les instructions de ces sous-algorithmes peuvent utiliser cette variable) ;
- Une variable définie au niveau de l'algorithme principal (celui qui permet de résoudre le problème initial, le problème de plus haut niveau) est appelée **variable globale** (sa portée est totale et couvre tout sous-algorithme de l'algorithme principal) ;

```
Algorithme P_Principale
Entier X  //Variable globale
Début
Sous_Algo1()   } La variable X est connue dans
Sous_Algo2();  } Sous_Algo1 et Sous_Algo2
Fin
```

- Une variable définie au sein d'un sous algorithme est appelée **variable locale** (sa portée est limitée au sous-algorithme qui la déclare).

Sous_Algo1 Entier X //Variable //locale Début ... Fin	Sous_Algo2 Entier Y //Variable //locale Début Y←X Fin
La variable X est inconnue au niveau de l'affectation Y←X du sous_Algo2	

- Lorsque le nom d'une variable locale est identique à une variable globale, la variable globale est localement masquée (Dans le sous-algorithme, la variable globale devient inaccessible).

Algorithme P_Principale
Entier X //Variable globale
Sous_Algo1 (X)
Entier X // Variable locale
Début
X←3
Fin
	Début
	X←5
	Sous_Algo1(X) ; //La variable //X prend La valeur 3
	Fin

5.2.6) Les paramètres formels/effectifs

Soit l'exemple qui permet de lire deux entiers n et p et d'effectuer le calcul combinatoire $C_n^p = \frac{n!}{(n-p)!*p!}$.

Nous remarquons que ce problème exige trois utilisations de la fonction factorielle que nous avons construite plus haut : factorielle(n), factorielle (n-p) et factorielle(p). Compte tenu de ses besoins, l'algorithme principal se prépare à utiliser la fonction factorielle avec des outils différents :

Utilisation 1 : Désire retourner la factorielle de n : (nb←n).

Utilisation 2 : Désire retourner la factorielle de n-p : (nb←n-p).

Utilisation 3 : Désire retourner la factorielle de p : (nb←p).

Les variables "n", "p" et "n-p" sont des paramètres. Cependant, la fonction factorielle est capable de s'exécuter en faisant jouer à (n, p et n-p) les rôles de nb. Ainsi, chacune de ces trois utilisations se résume par l'exécution des actions paramétrées comme suit :

Utilisation1 : factorielle (n)
Utilisation2 : factorielle (n-p)
Utilisation3 : factorielle (p)

Nous indiquons que chacune de ces trois actions ci-dessus est un appel à la fonction factorielle. Dans Utilisation1, la variable "**nb**" représente le **paramètre formel** alors que, "**n**" représente le **paramètre effectif** d'un appel de la fonction factorielle.

Définitions :

- Un paramètre formel est une variable choisie comme paramètre à la définition d'un algorithme.
- Un paramètre effectif est une variable ou une constante utilisée dans un appel à la place d'un paramètre formel.
- À l'exécution d'un appel, la correspondance entre un paramètre effectif et un paramètre formel est définie par la position, donc doivent être de même type.
- Un paramètre effectif est aussi appelé argument
- Un paramètre formel est aussi appelé paramètre
- Un passage par référence/adresse existe lorsque le paramètre formel subit un passage en sortie ou en entrée/sortie.
- Un passage par valeur se produit lorsque le paramètre formel subi un passage en entrée

Il existe trois types d'association (que l'on nomme passage de paramètre) entre le paramètre formel et le paramètre effectif du sous-algorithme appelant.

a) Le passage de paramètres en entrée

Dans ce type de passage, les instructions d'un sous-algorithme ne peuvent pas modifier le paramètre effectif. En fait, c'est la **valeur** du paramètre effectif qui est copiée dans le paramètre formel. Ce passage de paramètre est le seul qui admet l'utilisation d'une constante. Par exemple, le sous-algorithme

«*factorielle* » *qui* permet de calculer la factorielle d'un nombre admet un paramètre dont le passage est en entrée.

b) Les passage de paramètres en sortie

Avec ce type de passage, les instructions du sous-algorithme affectent obligatoirement la valeur du paramètre formel (valeur qui est aussi affectée au paramètre effectif). Il y a donc une liaison forte entre le paramètre formel et le paramètre effectif ; C'est pour cela qu'on ne peut pas utiliser des constantes pour ce type de paramètre (on utilise surtout des références). La valeur que pouvait posséder la variable utilisée comme paramètre effectif n'est pas utilisée par le sous-algorithme. Par exemple, le sous-algorithme **Lire** qui permet de mettre dans des variables des valeurs saisies par l'utilisateur admet n paramètres en sortie.

c) Le passages de paramètres en entrée/sortie

Ce type de passage combine les deux précédentes. Le passage de paramètre en entrée/sortie est adopté lorsque le sous-algorithme doit utiliser et/ou modifier la valeur de la variable du sous-algorithme appelant. Avec ce type de passage, on ne peut pas utiliser les constantes. Comme exemple d'un sous-algorithme, nous pouvons citer l'exemple de la procédure « **échanger** » permettant l'échange des valeurs de deux variables.

- **Application 1 :** Écrire *l'algorithme principal qui définit la procédure échange décrite plus haut.*

Solution :

Déclarations globales :
D.E : Entier a, b

```
Procédure echanger ( E/S Entier val1, E/S Entier val2)
 Déclarations locales :
  Entier aux
 Début
 aux←val1
 val1←val2
 val2←aux
 FinProc
 Début
   Ecrire ("Entrez deux entiers :")
   Lire (a,b)
   Echanger (a, b)
   Ecrire ("a=Entier et b =Entier", a, b)
Fin
```

Règles

- Lors de l'exécution de la procédure *echanger*, la variable *a* et le paramètre *val1* sont associés par un passage de paramètre en entrée/sortie : Toute modification sur val1 est effectuée sur a (de même pour b et val2).
- Par défaut si la déclaration d'un paramètre n'est pas précédée par E/S, E ou S, alors le passage de paramètre s'effectue, par défaut, en entrée (E).

Application 2 : *En utilisant le principe des paramètres modifiables, créer la fonction foisMoinSomme, qui calcule la somme et le produit de 2 nombres entiers, stocke et retourne la soustraction du produit par la somme.*

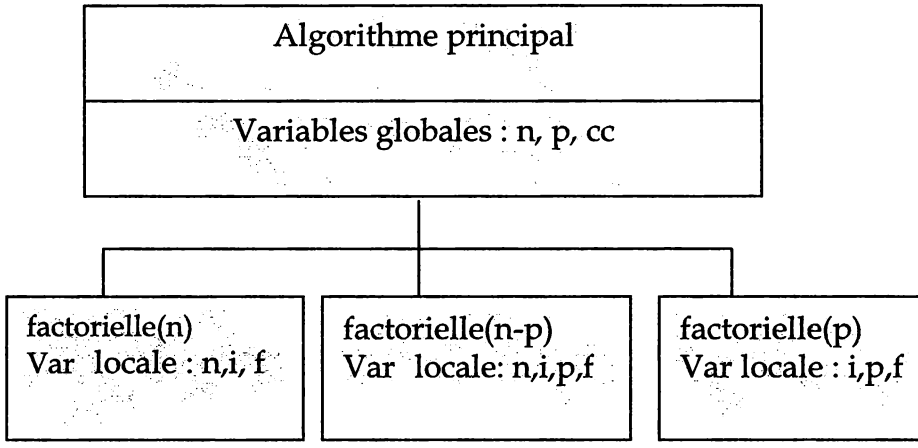
Solution :

```
Fonction Entier foisMoinSomme (Entier a, Entier b, S Entier s, S Entier p)
 Début
```

```
    s←a+b
    p←a*b
    Retourner (s-p)
Fin
```

d) Exemples de passage par paramètre

- **Application 1** : *Soit l'exemple permettant de faire le calcul combinatoire C_n^p. Ecrire l'algorithme principal permettant de résoudre ce problème.*



La résolution de ce problème s'effectue de la manière suivante :

- **Solution :**

```
Déclarations globales :
D.E : Entier n, p,
D.S : Réel cc
Fonction EntLong factorielle (Entier nb)
        //le type de retour est un entier long
EntLong f, i  //variables locales
Début
 f←1
Pour i de 1 à nb
```

```
Faire
f←f*i
FinPour
 Retourner (f)
FinFct
Début
Ecrire("Entrez un entier n et un entier p :")
Lire(n,p)
cc=factorielle(n)/(factorielle(n-p)*factorielle(p))
Ecrire ("Le résultat de la combinaison de Entier parmi Entier= Réel", p, n, cc)
Fin
```

Application 2 : *Écrire un algorithme qui affiche les nombres déficients (dont la valeur est plus grande que la somme de ses diviseurs) compris entre 0 et une valeur n saisie par l'utilisateur.*

Solution :

```
Variables globales
D.E : Naturel nb
Déclaration des Sous-algorithmes :
Fonction Naturel saisir () début...fin
Fonction Booléen estUnDiviseur (Naturel a,b) début...fin
Fonction Naturel calculerSommeDesDiviseurs (Naturel n) début...fin
Fonction Booléen estDéficient (Naturel n) début...fin
Procédure afficherNbsDéficients ( E Naturel max ) début...fin
Début
nb←saisir()
afficherNbsDéficients (nb)
Fin
```

Les définitions des sous-algorithmes déclarés dans l'algorithme principal se présentent comme suit :

1)

```
Fonction Naturel saisir ()
Naturel resultat
Début
 Ecrire("Valeur maximale d'affichage des nombres défi  cients") ;
Lire(resultat)
retourner resultat
FinFct
```

2)

```
  Fonction Booléen estUnDiviseur (Naturel a,b)
   Début
    Retourner (a % b=0)
   Fin
```

3)

```
Fonction Naturel calculerSommeDesDiviseurs (Naturel n)
     Naturel i, somme ;
    Début
     somme ←0
     Pour i  de 1 à  n/ 2 Faire
     Si (estUnDiviseur(n,i)=Vrai) Alors
      somme←somme+i
     Finsi
     FinPour
    Retourner (somme)
 Fin
```

4)

```
   Fonction Booléen estDéficient (Naturel n)
    Début
     Retourner (n>calculerSommeDesDiviseurs(n))
   Fin
```

5)

```
   Procédure afficherNbsDéficients (E Naturel max)
    Naturel i
    Début
    Pour i de 1 à max Faire
      Si (estParfait(i)) Alors
```

```
    Ecrire(i)
  Finsi
  Finpour
Fin
```

e) Circulation des informations dans un algorithme

Deux catégories d'information peuvent être circulées dans un algorithme : des informations transmises par l'algorithme principal aux sous-algorithmes ou des informations retournées par les sous-algorithmes à l'algorithme principal, comme présentée par la figure ci-dessous :

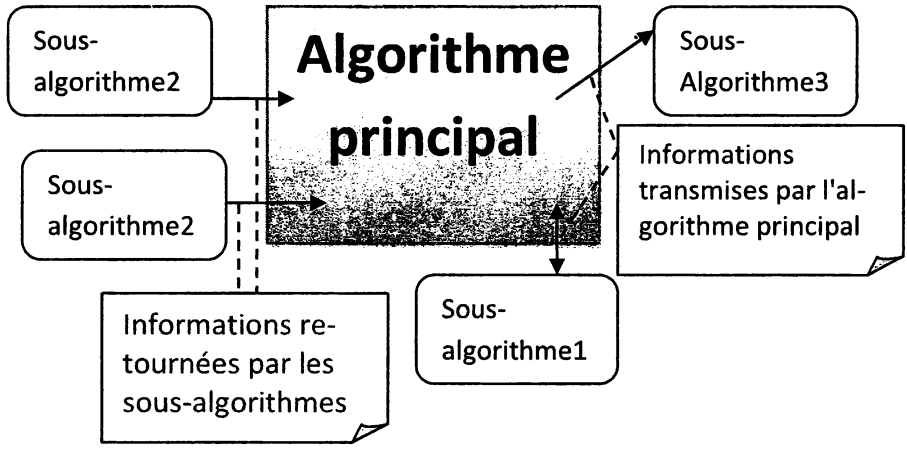


Figure 10. Echange de données entre un algorithme principal et ses sous-algorithmes

5.3) Les fonctions récursives

Une fonction récursive est une fonction qui fait appel à lui-même tant qu'une condition d'arrêt n'est pas atteinte. Elle permet de résoudre certains problèmes d'une manière très rapide, alors que si on devait les résoudre de manière itérative, il nous faudrait beaucoup plus de temps et de structures de données intermédiaires. Toute fonction récursive comporte une instruction (ou un bloc d'instructions) nommée "point terminal" ou "point d'appui" ou "point d'arrêt", qui indique que le reste des instructions ne doit plus être exécuté.

La récursivité utilise toujours la pile du programme en cours. On appelle "pile" une zone mémoire réservée à chaque algorithme ; sa taille peut être fixée manuellement par l'utilisateur. Son rôle est de stocker les variables locales et les paramètres d'une fonction. Dans une procédure récursive, toutes les variables locales sont stockées dans la pile, et empilées autant de fois qu'il y a d'appels récursifs. Donc, la pile se remplit progressivement, et si on ne fait pas attention on arrive à un "débordement de pile". Ensuite, les variables sont désempilées (dépilées). Il y a donc un "empilement" des espaces alloués aux variables locales, parallèlement à un "empilement" des appels de la fonction. Ce n'est que lors de l'exécution de la première instruction "Retourner" que l'on commence à "dépiler" les appels et les emplacements et donc à libérer de l'espace mémoire. La syntaxe d'une fonction récursive se définit comme suit :

Syntaxe de déclaration d'une fonction récursive

```
Fonction Type  Fonction_recursive(paramètres)
// Déclaration des variables locales
Début
  Si TEST_D'ARRET Alors
   <Instructions du point d'arrêt>
   Sinon  //----- exécution
   Débutsinon    <instructions>
   Fonction_recursive (paramètres changés);/
pel récursif
   <instructions>
    Finsinon
   Finsi
  FinFct
FinFct
```

- **Application 1 :** *Soit une version itérative d'une fonction qui fait le modulo de deux entiers a et b. Proposer une version récursive de cette fonction.*

Version itérative :

```
Fonction Entier modulo (Entier a, Entier b)
Début
TantQue (a>b)
 Faire
  a←a-b
 FinTantQue
 Retourner (a)
Fin
```

- **Solution :**

```
Fonction Entier modulo (Entier a, Entier b)
Début
```

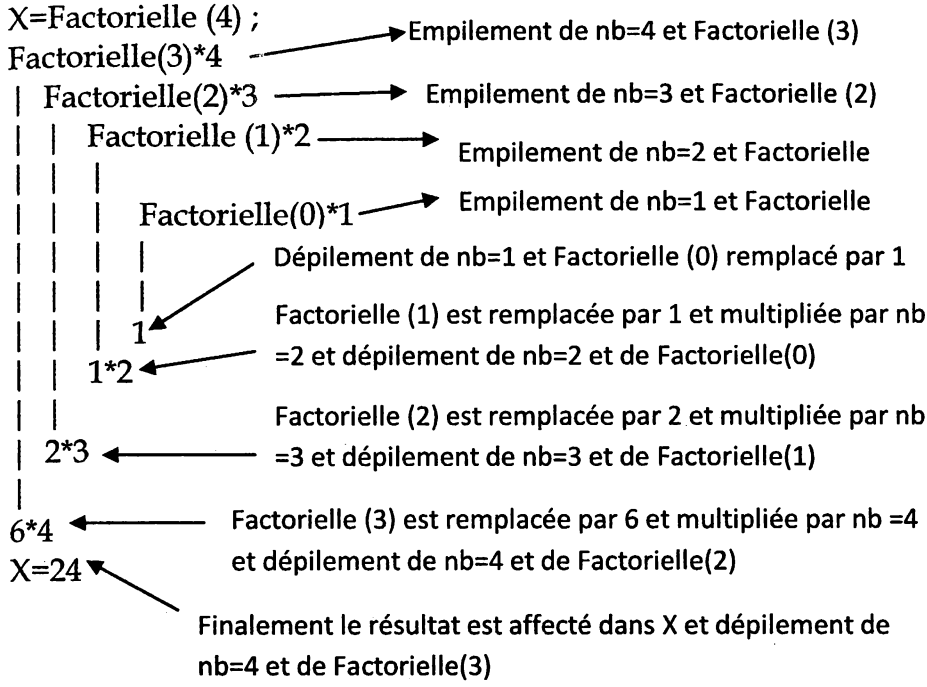
```
Si (a<=b) Alors
Retourner(a)
Sinon
Retourner (modulo(a-b,b))
Finsi
FinFct
```

Application 2 : *Ecrire une version récursive, au lieu de la version itérative de la fonction factorielle présentée plus haut.*

Solution :

```
Fonction Entlong factorielle (Entier nb)
  Début
   Si (nb≥1) Alors
    Retourner (factorielle (nb-1)*nb)
    Sinon //condition d'appui
    Retourner (1)
 FinFct
```

En effet, appliquons cette fonction sur un exemple : Nous étudions le résultat d'un argument=4, pour un appel récursif de la factorielle.


X=Factorielle (4) ;
Factorielle(3)*4
Empilement de nb=4 et Factorielle (3)
Factorielle(2)*3
Empilement de nb=3 et Factorielle (2)
Factorielle (1)*2
Empilement de nb=2 et Factorielle
Factorielle(0)*1
Empilement de nb=1 et Factorielle
Dépilement de nb=1 et Factorielle (0) remplacé par 1
1
Factorielle (1) est remplacée par 1 et multipliée par nb =2 et dépilement de nb=2 et de Factorielle(0)
1*2
Factorielle (2) est remplacée par 2 et multipliée par nb =3 et dépilement de nb=3 et de Factorielle(1)
2*3
6*4
Factorielle (3) est remplacée par 6 et multipliée par nb =4 et dépilement de nb=4 et de Factorielle(2)
X=24
Finalement le résultat est affecté dans X et dépilement de nb=4 et de Factorielle(3)

Exercices du chapitre 5

Exercice 5.1

a) Ecrire les sous-algorithmes MIN et MAX qui déterminent et retournent le minimum et le maximum de deux nombres réels.

b) Ecrire un algorithme principal se servant des sous-algorithmes MIN et MAX pour déterminer le minimum et le maximum de quatre nombres réels saisis par un utilisateur.

Exercice 5.2

Soit l'algorithme permettant de trier trois nombres saisis par l'utilisateur :

```
D.E : Entiers a, b, c
D.I: Caractère encore
Début
encore← réponseUtilisateur()
TantQue (encore='o')
 Faire
  Ecrire("***Premier nombre***")
  demanderNombre(a)
   Ecrire("***Deuxième nombre***")
   demanderNombre(b)
   Ecrire("***Troisième nombre***")
   demanderNombre(c)
   classerDeuxNombres(a, b)
  classerDeuxNombres(b, c)
 classerDeuxNombres(a, b)
Ecrire ("Voici vos nombres rangés dans l'ordre croissant : Entier, Entier, Entier", a, b, c)
```

encore ← réponseUtilisateur()
FinTantQue
Fin

a) Quels sont les sous-algorithmes utilisés par cet algorithme ? Donner la spécification de ces sous-algorithmes (fonction/procédure, liste de paramètres et leurs types).
b) Définir les sous-algorithmes détectés.
c) Simuler l'exécution de l'algorithme sur les données suivantes : 3, 2, 1.

Exercice 5.3

Soit le sous-algorithme suivant :

```
Fonction Entier fct (int n) Début
  Entier k←0
 TantQue (n>0) Faire
  k←k+1
   n←n/10
  FinTantQue
Retourner (k)
FinFct
```

Choisir pour chacune des propositions suivantes celle qui désigne la réponse correcte.
1) Quand on appelle la fonction fct avec comme paramètre un entier positif n, elle renvoie :
   (a) la somme des chiffres de n
   (b) le nombre de chiffres de n
   (c) le nombre de chiffres non nuls de n
2) Quand on appelle la fonction fct avec comme paramètre un entier strictement négatif n, elle :
   (a) plante (provoque l'arrêt de l'algorithme)

(b) ne s'arrête jamais

(c) renvoie 0

3) On remplace la ligne 3 par **TantQue (n>=0) Faire**. La nouvelle fonction fct :

(a) fonctionne exactement comme avant

(b) ne s'arrête jamais quand son paramètre est positif ou nul (et fonctionne comme avant dans les autres cas)

(c) fonctionne exactement comme avant sauf quand son paramètre est nul (auquel cas la méthode ne s'arrête jamais)

Exercice 5.4

Ecrire un sous-algorithme qui reçoit en paramètre deux entiers n et i et retourne le nombre de fois où le chiffre i apparaît dans le nombre n. Par exemple, si n vaut 12332 et si i vaut 3, on obtient comme résultat 2. Proposer deux solutions comme suit :

1) version itérative
2) version récursive

Exercice 5.5

Ecrire un sous-algorithme de saisie contrôlée prenant comme paramètres deux réels a et b. Ce sou-algorithme doit renvoyer un réel saisi par l'utilisateur et appartenant à l'intervalle [a, b]. Si l'utilisateur ne fournit pas un réel valide, le sou-algorithme doit recommencer la saisie.

Exercice 5.6

a) Ecrire une fonction **continuer (Rép)** qui retourne VRAI si le caractère passé en paramètre est égal à 'o' ou 'O', et FAUX sinon.
b) Ecrire une procédure **calcul (annNaiss, annCour, age)** qui calcule l'âge qu'aura une personne née durant l'année *annNaiss* à la fin de l'année courante *annCour.*
c) Ecrire un algorithme **age** qui demande à l'utilisateur l'année de naissance d'un individu, puis qui calcule et affiche l'âge de ce dernier à la fin de l'année courante (l'année courante sera fournie par l'utilisateur). Cet algorithme doit utiliser les sous-algorithmes précédents. L'utilisateur doit pouvoir continuer à fournir des années de naissance aussi longtemps qu'il le désire.

Exercice 5.7

Cet exercice demande de donner la résolution des énoncés pour les sous-algorithmes et algorithmes correspondants comme présenté dans les points suivants :

a) Une fonction **estUnChiffre** qui reçoit un nombre entier et qui retourne VRAI si ce nombre est un chiffre, et FAUX sinon.
b) Une fonction **donneNombre** qui reçoit 2 chiffres et qui retourne le nombre obtenu avec ces 2 chiffres (par exemple, si les chiffres sont 2 et 3, on retourne le nombre 23).
c) Une fonction **somme1aN** qui reçoit un nombre en paramètre et qui retourne la somme des entiers de 1 jusqu'à ce nombre.
d) Une procédure **minMax** qui saisit nbVal valeurs (nbVal est un paramètre de la fonction) et qui renvoie (par le biais des paramètres) le mininum, le maximum et la moyenne des valeurs saisies. Si le nombre nbVal donné est nul, on traitera ce cas d'erreur en levant une exception.

e) Un algorithme qui demande à l'utilisateur d'entrer deux nombres entiers qui soient des chiffres. Une fois la saisie validée, l'algorithme doit afficher le nombre **unNombre** obtenu à partir de ces deux chiffres, puis afficher la somme des entiers de 1 jusqu'à **unNombre.** Ensuite, l'algorithme doit demander à l'utilisateur de saisir un autre nombre entier positif **nbVal**, puis saisir **nbVal** valeurs et afficher la plus grande, la plus petite et la moyenne des valeurs saisies. Vous devrez bien sûr faire appel aux sous-algorithmes définis en a), b), c) et d).

Exercice 5.8

L'objectif de cet exercice est de rédiger un algorithme pour vérifier si deux entiers strictement positifs sont amis ou non. Deux entiers x et y (>0) sont dits nombres amis si et seulement si :

- La somme des diviseurs de x sauf lui-même est égal à y
- La somme des diviseurs de y sauf lui-même est égal à x

Exemple : 220 et 284 sont deux nombres amis

1) Avant de rédiger l'algorithme principal, créer les sous-algorithmes suivants :
   a) Le sous-algorithme saisie ()
   b) Le sous-algorithme somme_diviseur (), ce sous-algorithme reçoit comme paramètre un nombre entier>0 et retourne la somme de ses diviseurs.
2) Déduire l'algorithme principal

Exercice 5.9

Un nombre entier est dit «palindrome» s'il peut se lire de gauche à droite et de droite à gauche en représentant le même nombre. Les nombres suivants sont palindromes : 825528, 525, 3.

a) Ecrire une fonction **extrait** qui reçoit un entier (nb) en paramètre et qui :
- retourne -1 et ne modifie pas nb si nb<10
- retourne le 1er chiffre de nb et modifie nb en retirant le chiffre de plus fort poids. Par exemple extrait (4657) retourne 4 et nb vaut au retour 657

b) Ecrire, en utilisant la fonction précédente une fonction «booléenne» estPalindrome qui reçoit un entier en paramètre et retourne vrai ou faux selon que cet entier est un palindrome ou non ;

c) Ecrire un algorithme principal qui teste la première puis la deuxième fonction afin de vérifier si un nombre entier est palindrome ou non.

Exercice 5.10

Nous voulons gérer ici, le problème d'affichage du montant d'une facture en toutes lettres, sachant que ce montant est un entier compris entre 0 et 9999. Par exemple, pour M=2010, le programme doit afficher "deux milles dix'.

Note :

- La valeur 11 doit s'afficher "onze" et non pas "dix-un". De même 12, 13, etc. Donc, les valeurs de 0 à 19 sont à considérer comme des unités.
- Le montant 21 s'affiche "vingt et un" alors que 22, 23 etc. s'écrivent "vingt deux", "vingt trois", etc. De même pour les montants 31, 41, 51, 61 et 71, on affichera "trente et un",..., "soixante et onze".
- Le montant 128 s'écrit "cent vingt huit" et non pas "un cent vingt huit".

Pour résoudre ce problème, nous proposons de commencer par écrire les cinq procédures suivantes :

a) Ecrire la procédure qui doit déduire d'un nombre N compris entre 0 et 9999, les différentes chiffres : Les milliers, les centaines, les dizaines et les unités.

Exemple : Pour N=3297
Millier=3
Centaine=2
Dizaine=9
Unité=7

b) Ecrire la procédure affiche_unité qui reçoit en paramètre un nombre compris entre 1 et 19 et affiche comme résultat ce nombre en toutes lettres.

c) Ecrire la procédure affiche_dizaine qui a pour donnée un nombre compris entre 0 et 9 et affiche comme résultat le chiffre en toutes lettres, sachant qu'elle n'affiche rien pour le chiffre 0 et 1. En plus, cette procédure affiche le même résultat : "soixante" pour les deux chiffres 6 et 7 ; et pour les chiffres 8 et 9 affiche le même résultat "quatre-vingt".

d) Ecrire la procédure affiche_centaine qui reçoit en paramètre un nombre compris entre 1 et 9, si le nombre est égale à 1 elle affiche "mille" et non pas "un mille", sinon elle affiche "deux milles", " trois milles", etc...

e) Ecrire l'algorithme principal qui affiche le montant saisi d'une facture donnée (entier compris entre 0 et 9999) en toutes lettres, en faisant appel aux 5 procédures précédentes.

Exercice 5.11

Ecrire une fonction chaotique, qui calcule le nième terme de la suite définie récursivement par :

Q[1] = Q[2] = 1

Q[n] = Q[n - Q[n-1]] + Q[n - Q[n-2]] pour n >2

Sa définition rappelle celle de Fibonacci, mais chaque nouvelle valeur est calculée en regardant les deux valeurs immédiatement précédentes afin de savoir combien de nombres il faut reculer pour trouver les valeurs qui doivent être additionnées !

Voici les 17 premiers termes : 1, 1, 2, 3, 3, 4, 5, 5, 6, 6, 6, 8, 8, 8, 10, 9, 10. Le comportement de cette suite est très surprenant : plus on avance, plus elle paraît insensée, et le chaos semble y être produit de manière très ordonné.

Exercice 5.12

La fonction récursive suivante permet d'effectuer la multiplication de deux entiers positifs non nuls p et q en utilisant uniquement l'addition et la soustraction :

$$\begin{cases} ProdRec(p, q) = ProdRec(p-1, q)+q & \text{si } p \geq 1 \\ ProdRec(1, q) = q \end{cases}$$

Écrire la définition récursive de la fonction ProdRec qui reçoit en entrée deux entiers p et q et renvoie la valeur du produit p × q.

Exercice 5.13

Ecrire une fonction récursive qui vérifie et indique si deux nombres entiers positifs non nuls n et m sont premiers entre eux. Soit à considérer les indications suivantes :

- Si n ou m vaut 1 alors n et m sont premiers entre eux
- Si n=m et n<>1 alors ils ne sont pas premiers entre eux
- Si n et m sont pairs, ils ne sont pas premiers entre eux

- Si n est pair et m impair alors n et m sont premiers entre eux si et seulement si n/2 et m le sont (idem en inversant les rôles de m et n)
- Si n et m sont impairs et n>m alors n et m sont premiers entre eux si et seulement si (n-m)/2 et m le sont (idem en inversant les rôles de n et m).

Exercice 5.14

Ecrire une fonction récursive qui détermine le chiffre au k^{éme} position à partir de la droite d'un entier n>0 :

Indication :

Le 3^{éme} chiffre à partir de la droite de 6752 est 7

Le 5^{éme} chiffre à partir de la droite de 6752 est 6

Corrigé des exercices du chapitre 5

Exercice 5.1 (Solution)

a)

```
Fonction Réel Max
(Réel a, Réel b)
Début
Si (a>b)  Alors
Retourner(a)
 Sinon
 Retourner(b)
FinSi
FinFct
```

```
Fonction Réel
Min (Réel a, Réel b)
Début
Si (a<b) Alors
Retourner(a)
Sinon
Retourner(b)
Finsi
FinFct
```

b)

```
D.E : Réel a, b, c, d
D.S : Réel Maxm, Minm
Début
Ecrire ("donner 4 réels")
Lire (a, b, c, d)
Maxm ←Max (Max (a, b), Max(c, d))
Ecrire ("le Maximum est = Réel", Maxm)
Minm←Min (Min (a, b), Min(c,d))
Ecrire ("le Minimum est = Réel", Minm)
Fin
```

Exercice 5.2 (Solution)

a) Les sous algorithmes utilisés par cet algorithme sont :

- réponseUtilisateur
- demanderNombre
- classerDeuxNombres

➡ Spécification :

- Fonction Caractère réponseUtilisateur ()
- Procédure demanderNombre (E/S Entier n)
- Procédure classerDeuxNombres *(E/S Entier a, E/S Entier b)*

b)

```
Fonction Caractère réponseUtilisateur ()
 Caractère rep
Début
Répéter
  Ecrire ("voulez vous continuer? O/N")
  Lire(rep)
Jusqu'à (rep='o' OU rep ='O' OU rep='n' OU rep='N')
 Retourner (rep)
FinFct

Procédure  demanderNombre (E/S Entier n)
 Début
 Ecrire ("Donner un entier ")
 Lire (n);
FinProc

Procédure classerDeuxNombres  (E/S  Entier a, E/S  Entier b)
 Entier aux
Début
```

Si *(a>b)* **Alors**
aux←a
a←b
b←aux
FinSi
FinProc

c)

a	b	c	Encore
3	2	1	O
2	1	3	n
1	2		

➡ Affichage de l'algorithme :
Premier nombre 3
Deuxième nombre 2
Troisième nombre 1

Voici vos nombres rangés dans l'ordre croissant : 1, 2, 3

Exercice 5.3 (Solution)

1) La bonne réponse est b)

2) La bonne réponse est c)

3) La bonne réponse est b)

Exercice 5.4 (Solution)

1) version itérative

```
Fonction Entier nbchcont (Entier n, Entier i)
Entier cpt
Début
  cpt←0
  Répéter
    Si (n%10=i) Alors
      cpt←cpt+1
    Finsi
   n←n/10
  Jusqu'à (n=0)
Retourner (cpt)
FinFct
```

2) version récursive

```
Fonction Entier nbchcont (Entier n, Entier i)
Entier cpt
Début
  Si (n=0) Alors
   Retourner (0)
   Sinon
    Si (n%10=i) Alors
      Retourner (1+nbchcont (n/10, i))
    Sinon
     Retourner (nbchcont (n/10, i))
  Finsi
Finsi
FinFct
```

Exercice 5.5 (Solution)

Fonction Réel saisieContrôlee (Réel a, Réel b)
Réel x
Début
Répéter
  Ecrire ("donner un réel")
  Lire (x)
Jusqu'à (E(x)<>x **ET** x>a **ET** x<b)
Retourner (x)
Fin

- E est une fonction qui désigne la partie entière d'un nombre dont la définition est comme ci-dessous :

Fonction Entier E (**Réel** x)
Entier pe
Début
  pe←0
  TantQue (pe≤x)
   Faire
    pe←pe+1
   FinTantQue
  Retourner (pe-1)
FinFct

Exercice 5.6 (Solution)

a)

Fonction Booléen continuer (Caractère c)
Début
Si ((c='o') ou (c='O')) **Alors**

```
 Retourner (vrai)
 Sinon
 Retourner (Faux);
FinSi
FinFct
```

b)

```
Procédure calcul(E Entier an, ac, S Entier ag)
Début
 ag←ac-an
FinProc
```

c)

```
D.E: Entier an, ac, s
      Caractère c
D.S : Entier ag
Début
 Répéter
  Ecrire (" donner l'année de naissance") ;
  Lire (an) ;
   Ecrire ("donner l'année courante aaaa") ;
   Lire (ac);
    calcul (an, ac, age);
   Ecrire ("l'age est un entier ", age);
  Ecrire ("Voulez vous continuer ? o|O|n|N") ;
 Lire (c)
Jusuq'à (continuer(c)=Faux)
Fin
```

Exercice 5.7 (Solution)

a)

```
Fonction booléen estUnChiffre (E Entier n)
Début
 Si ((n>=0) et n<10)) Alors
  Retourner (Vrai)
 Sinon
  Retourner (Faux)
Finsi
FinFct
b)
Fonction Entier donneNombre (a, b)
Retourner (a*10+b)
FinFct
c)
Fonction Entier somme1aN (Entier n)
Entier i, s
Début
s←0
Pour i de 1 à n Faire
 s←s+i
Fin pour
Retourner (s)
FinFct

d)
Procédure minMax (E Entier nbval, S Réel moy, min, max)
Réel nb ; Entier i
Début
 Si (nbval<>0) Alors
  Débutalors
    max←0
    min←0
    moy←0
```

```
 Pour i de 1 à nbval
  Faire
   Ecrire ("Donner un réel")
   Lire (nb)
   moy← moy+(nb div nbval)
   Si(max<nb) Alors
    max←nb
   FinSi
   Si(min>nb) Alors
    min←nb
   FinSi
  FinPour
 Finalors
 Sinon
 Ecrire ("Erreur de saisie : Il faut saisir un nombre de valeurs>0")
 FinSi
Fin Proc
```

e)

```
D.E : Entier a, b, nbVal
D.S : Entier x, som
      Réel moy, min, max
Début
 Répéter
  Ecrire ("Donner deux chiffres a et b") ;
  Lire (a, b)
 Jusqu'à (estUnChiffre(a)=Vrai ET estUnChiffre(b)=Vrai)
 x← donneNombre (a, b)
 Ecrire ("Entier concatené avec Entier =Entier",a,b,x)
 som← somme1aN (x)
```

```
 Ecrire ("La somme des Entier premiers entiers est =Entier",x,
som)
 //saisie de nbval valeurs
 Ecrire ("le nombre des réels à saisir")
 Lire (nbVal)
 minMax(nbVal, moy, min, max) ;
 Ecrire ("la valeur minimale est =Réel", min) ;
 Ecrire ("la valeur maximale est = Réel", max) ;
Ecrire ("la moyenne des valeurs saisis est = Réel", moy) ;
Fin
```

Exercice 5.8 (Solution)

1)

a)

```
Procédure saisie (Entier x, Entier y)
 Début
  Répéter
   Lire (x, y)
  Jusqu'à (x>0 ET y>0)
FinProc
```

b)

```
Fonction Entier somme_diviseur (Entier n)
 Entier i, sd
 Début
  sd←1
 Pour i de 2 à n/2
  Si (n%i=0) Alors
   sd←sd+i
  Finsi
```

```
 FinPour
 Retourner (sd)
FinFct
```

2)

```
D.E : Entier x, y
Début
 saisie (x, y)
  Si (somme_diviseurs (x)=y ET somme_diviseurs (y)=x)
    Alors
     Ecrire (''Entier et Entier sont des nombres amis'', x, y)
     Sinon
     Ecrire (''Entier et Entier ne sont pas des nombres amis'', x,
y)
  Finsi
 Fin
```

Exercice 5.9 (Solution)

a)

```
Fonction Entier extrait (Entier nb, E/S Entier cpt)
Début
 cpt←0 ;
 Si (nb>0 ET nb<10) Alors
  Retourner (-1)
  Sinon
   Debutsinon //puisqu'il s'agit d'un nombre d'instructions >1
   TantQue (Non  estChiffre (nb))
   Faire
    nb←nb/10
    cpt←cpt+1
   FinTantQue
```

```
   Retourner(nb)
  FinSinon
FinFct

Fonction booléen estChiffre (Entier n)
Début
 Si ((n>=0) et n<10)) Alors
  Retourner (Vrai)
 Sinon
  Retourner (Faux) ;
Finsi
FinFct
```

b)

```
Fonction Booléen estPalindrome (Entier n)
Entier cpt ;
Entier c ;
Debut
 TantQue ((c←extrait(n, cpt))=n%10 ET  n <>0)
  Faire
   n←(n-c*(10^cpt))/10
  FinTantQue
  Si (n=0)  Alors
   Retourner (vrai)
   Sinon
   Retourner (Faux)
 Finsi
FinFct
```

c) Algorithme principal

```
D.E : Entier n, cpt
Début
  Ecrire (''donner n'')
```

```
Lire (n)
Si (extrait(n, , cpt) <>-1) Alors
Si (estPalindrome(n)=vrai)   Alors
 Ecrire (''Entier c'est un palindrome'', n)
 Sinon
 Ecrire (''Entier ce n'est pas un palindrome'', n)
Finsi
Sinon
Ecrire (''Le nombre doit être composé de plus qu'un chiffre'')
Fin
```

Exercice 5.10 (Solution)

a)

```
Procedure affich_chiff (Entier  n, S Entier m, S Entier c, S Entier d, S Entier u)
Debut
  m←n/1000
  c← (n/100)%10
  d← (n/10)%10
  u←n%10
Fin Proc
```

b)

```
Procedure affich_unit (Entier n)
Debut
Selon (n)
Faire
  cas 1 : Ecrire ("un")
  cas 2 : Ecrire ("deux")
  cas 3 : Ecrire ("trois")
  cas 4 : Ecrire ("quatre")
```

```
  cas 5 : Ecrire ("cinque")
  cas 6 : Ecrire ("six")
  cas 7 : Ecrire ("sept")
  cas 8 : Ecrire ("huit")
  cas 9 : Ecrire ("neuf")
  cas 10 : Ecrire ("dix")
  cas 11 : Ecrire ("onze")
  cas 12 : Ecrire ("douze")
  cas 13: Ecrire("treize")
  cas 14 : Ecrire("quatorze")
  cas 15 : Ecrire("quinze")
  cas 16 : Ecrire("seize")
  cas 17 : Ecrire("dix sept")
  cas 18 : Ecrire("dix huit")
  cas 19 : Ecrire("dix neuf")
Sinon : Ecrire(" ")
Finselon
Finproc
```

c)

```
Procedure affich_diz (Entier c)
Debut
 Selon (c)
 Début
  cas 2 : Ecrire ("vingt")
  cas 3 : Ecrire ("trente")
  cas 4 : Ecrire ("quarante")
  cas 5 : Ecrire ("cinquante")
  cas 6, 7 : Ecrire ("soixante")
  cas 8, 9 : Ecrire ("quatre_vingt")
 Sinon : Ecrire (" ")
```

```
 Finselon
 FinProc
d)
 Procedure affich_cent (Entier c)
 Debut
 Selon (c)
  Faire
   cas 1 : Ecrire ("mille")
   cas 2 : Ecrire ("deux milles")
   cas 3 : Ecrire ("trois milles")
   cas 4 : Ecrire ("quatre mille")
   cas 5 : Ecrire ("cinq mille")
   cas 6 : Ecrire ("six mille")
   cas 7 : Ecrire ("sept mille")
   case 8 : Ecrire ("huit mille")
    case 9 : Ecrire ("neuf mille")
  Sinon: Ecrire(" ")
   Finselon
  FinProc
d)
D.E : Entier n,x,y,z,t,u,d
//Déclaration des prototypes des sous-algorithmes
procédure affich_chiff(Entier, S Entier, S Entier, S Entier, S
Entier)
Procedure affich_unit(Entier)
Procedure affich_cent(Entier)
Procedure affich_diz(Entier)
Début
Répéter
Ecrire("donner un nombre")
Lire(n)
```

```
Jusqu'à (n>0 ET n<9999)
 affich_chiff(n, x, y, z, t)
 Ecrire ("le montant de la facture est: RC") // RC c.à.d. retour à
//la ligne
Si (n<20) Alors
  affich_unit(n)
Sinon
Si(n<100) Alors
 Si(t=0) Alors
 affich_diz(z)
  sinon
 Debsion
 affich_diz(z)
 Ecrire(" ")
 affich_unit(t)
  Finsinon
  Finsi
   Sinon
     Si (n<1000) Alors Debalors
      affich_cent(y)
       Ecrire(" ")
  affich_diz(z)   Ecrire(" ")
 affich_unit(t)  Finalors
 Sinon
DebSinon
affich_millier(x)  Ecrire(" ")
 affich_cent(y)   Ecrire(" ")
  affich_diz(z)   Ecrire(" ")
 affich_unit(t)  Finsinon
Finsi
Finsi
```

Finsi
Finsi
Finsi
Fin

Exercice 5.11 (Solution)

Fonction Entier chaotique (Naturel n)
Début
Si (n=2 OU n=1) **Alors**
Retourner (1)
Sinon
Retourner chaotique (n - chaotique (n-1)) + chaotique (n - chaotique (n-2))
Finsi
FinFct

Exercice 5.12 (Solution)

Fonction Entier ProdRec (**Entier** p, **Entier** q)
Début
Si (p=1) **Alors**
Retourner (q)
Sinon
Retourner (ProdRec(p − 1, q)+q)
FinSi
Fin

Exercice 5.13 (Solution)

Fonction Booléen estPremierEntreEux (Entier n, Entier m)
Début

```
Si (n=1 Et m=1) Alors
Retourner (Vrai)
Sinon
Si (n=m Et n<>1) Alors
 Retourner (Faux)
 Sinon
 Si (n%2=0 ET m%2<>0) Alors
   Retourner (estPremierEntreEux (n/2, m))
     Sinon
     Si (n%2=0 ET m%2=0) Alors
      Retourner (Faux)
      Sinon
      Si (n%2<> 0 ET m%2=0) Alors
      Retourner (estPremierEntreEux (n, m/2))
       Sinon
       Si (n%2= 0 ET m%2<>0) Alors
       Retourner (estPremierEntreEux (m, n/2))
       Sinon
      Si (n%2<>0 ET m%2<>0 ET n>m) Alors
      Retourner (estPremierEntreEux (n-m)/2, m)
       Sinon
        Si (n%2<>0 ET m%2<>0 ET m>n) Alors
        Retourner (estPremierEntreEux (m-n)/2, n)
        Sinon
         Retourner(Faux)
   Finsi
  FinSi
 Finsi
Finsi
Finsi
FinFct
```

Exercice 5.14 (Solution)

```
Fonction Entier chiffreposition (Entier n, position p)
  Début
   Si(p=1)
    Retourner (n%10)
    Sinon
    Retourner(n/10, p-1)
   FinSi
FinFct
```

Chapitre 6

La structure de données tableau

6.1) Utilité des tableaux 213

6.2) Tableau statique à une dimension 215

6.3) Opérations élémentaires sur les tableaux 216

6.3.1) Utilisation d'un tableau par les indices 216

6.3.2) Initialisation d'un tableau 216

6.3.3) Affichage des éléments d'un tableau 217

6.3.4) Ajout d'un élément à la fin d'un tableau 218

6.3.5) Recherche d'un élément dans un tableau 219

6.3.6) Recherche de la position d'un élément dans un tableau 220

6.3.7) Suppression d'un élément dans un tableau 221

6.3.8) Insertion d'un élément dans un tableau 222

6.3.9) Inversion de l'ordre des éléments d'un tableau 224

6.4) Applications sur les ensembles 225

6.5) Les tableaux bidimensionnels 231

CHAPITRE 6

□□□

La structure de données tableau

Résumé : **L'utilisation de la structure de données tableau facilite énormément le travail de mémorisation et de stockage de données. En effet, l'utilisation d'un tableau diminue le nombre des variables à déclarer, ainsi que le nombre de traitement à effectuer. L'énorme avantage des tableaux, c'est la possibilité de les traiter à travers les boucles. Les paragraphes qui suivent présentent plusieurs façons de déclaration des tableaux d'entier. En plus, décrivent une panoplie de manières permettant la recherche, l'initialisation et l'affichage des éléments d'un tableau à travers des algorithmes.**

6.1) Utilité des tableaux

Dans le cas de la manipulation de nombreuses variables, il devient vite pénible de devoir les déclarer toutes d'une manière séparée. Par exemple, dans le cas de l'utilisation de 20 variables entières pour calculer la moyenne de 20 notes ; Il est inutile de déclarer individuellement 20 variables entières : **Entier** e1, e2,

e3, ..., e20. Mais plutôt, il faut déclarer un tableau de 20 nombres entiers (**tableau d'entier e[20]**), accessibles par e[1], e[2], e[3], ... , e[20] qui permet de **rassembler toutes ces variables en une seule**, au sein de lesquelles chaque valeur sera désignée par un numéro (indice). Les nombres 1 à 20 désignent les indices (valeurs entières) du tableau.

Définitions :

Un tableau ou encore variable indicée représente un ensemble d'éléments de même type désigné par un identificateur unique, stocké de manière contiguë en mémoire et différentié par un système d'indices. Chaque élément contenu dans le tableau est repéré par un "indice" précisant sa position au sein de l'ensemble. Le nombre qui sert à repérer chaque valeur (e[i]) s'appelle l'élément d'indice (i).

Chaque fois qu'on voulait désigner un élément du tableau, on fait figurer le nom du tableau, suivi de l'indice de l'élément entre crochets.

Selon le problème, il existe 2 grands cas d'utilisation des tableaux : le cas où le nombre de variables à utiliser est grand et connu (cas des tableaux statiques) et l'autre cas plus fréquent, qui est adopté lorsque le nombre de variables est large, mais non précisément connu (cas des tableaux dynamiques).

Notes : Un tableau, c'est une zone mémoire où les éléments seront placés à des adresses correspondantes à leurs rangs dans la séquence. Ainsi, en codant un entier sur 4 octets, les éléments d'un tableau {23, 12, 34, 56, 3} seront codés sur 4*5 octets : si le premier élément 23 est à l'adresse 23002101, les éléments 12, 34, 56 et 3 seront aux adresses respectives 23002105, 23002109, 23002113 et 23002117. La conséquence pratique de ce choix est de

pouvoir accéder à un élément à partir de son rang (indice). En algorithmique, les indices démarrent à partir de la valeur 1, mais en C le démarrage s'effectue à partir de l'indice 0.

6.2) Tableau statique à une dimension

Un tableau à une dimension correspond à la notion de vecteur en mathématiques où la taille est fixée dès le début. La syntaxe utilisée pour déclarer des objets de type tableau statique est la suivante :

- **Déclaration1 :** Type_Elem nom_tableau:tableau [taille_Max_tab]
- **Déclaration2 :** Type_Elem nom_tableau:tableau [indice_Min, indice_Max]

Nous présentons quelques exemples de déclarations de tableaux, selon le besoin :

- **Entier** tab0:tableau[1..20] //tableau de 20 entiers non initialisé
- **Entier** tab1:tableau[20]
- **Entier** tab2:tableau[] ← {1,2,3,4,5} //tableau de 5 entiers
- **Entier** tab3:tableau [20] ← {1,2,3,4,5}//tableau de 20 entiers //partiellement initialisés
- **Type Entier** Tab:tableau [5] //définition du type Tab de taille 5 entiers
- Tab tab5
- Tab tab6 ← {1,2,3,4,5}

6.3) Opérations élémentaires sur les tableaux

Dans ce chapitre, on va exploiter les sous-algorithmes du chapitre précédent afin de présenter quelques opérations sur les tableaux. Comme le tableau ne contient pas sa propre taille, notre stratégie consiste à passer en paramètres la taille effective, ainsi que la taille physique. L'illustration de ces manipulations est faite sur des tableaux d'entiers.

6.3.1) Utilisation d'un tableau par les indices

Un tableau peut être accédé en lecture et/ou en écriture :

- **Accès en écriture : Ecrire** (tabl[i]) → {la valeur contenue dans l'indice i du tableau tabl est affichée à l'écran}
- **Accès en lecture :**
  - tabl[i] ←18 : La valeur 18 est placée dans l'indice i du tableau tabl.
  - Lire (tabl[i]) : La valeur entrée par l'utilisateur est enregistrée dans l'indice i du tableau tabl.

6.3.2) Initialisation d'un tableau

Application : *Ecrire un algorithme qui permet d'initialiser un tableau de dimension 1 à partir de valeurs lues au clavier et qui délivre le nombre de valeurs effectivement lues. La lecture se termine lorsque le nombre maximal de données pouvant être contenues dans le tableau est atteint.*

Solution

Constantes :

```
Entier TailleMAX←100
D.E : Entier nbVal, ind, nombres : tableau
[1..TailleMAX]

Début
 Répéter
   Ecrire ("Combien de valeurs voulez vous saisir ?")
   Lire (nbVal)
   {Refuser la saisie : la capacité du tableau est dépassée}
   Si (nbVal>TailleMax) Alors
    Ecrire ("trop de valeurs à saisir")
   Finsi
  Jusqu'à (nbVal <= TailleMAX)
  Pour ind de 1 à nbVal Faire
   Ecrire ("Donner une valeur") ;
    {Valeur à ranger dans la indème case du tableau}
   Lire (nombres[ind])
  Finpour
Fin
```

6.3.3) Affichage des éléments d'un tableau

Un tableau de taille n contient des éléments auxquels on peut accéder en faisant varier un indice de 1 à n. Soit la procédure *"afficheTableauEntiers"* permettant d'afficher les éléments d'un tableau tab.

```
//taillemax est une constante globale déclarée dans l'algorithme
principal
//taille est la taille effective du tableau
Procédure afficheTableauEntiers(Entier tab:tableau[taillemax],
Entier taille)
  Entier i
  Début
  Pour i de 1 à taille
```

```
  Faire
  Ecrire(tab[i])
 FinPour
FinProc
```

- Soit le tableau tab dont taillemax=taille=6

10	14	5	20	16	1

- Affichage après l'appel de la fonction afficheTableauEntiers : 10 14 5 20 16 1

6.3.4) Ajout d'un élément à la fin d'un tableau

L'ajout d'un élément se fait à la fin du tableau et vient occuper le premier emplacement libre, s'il existe. Dans ce cas, la taille effective du tableau est augmentée de 1. Il s'agit de créer une fonction qui prend en paramètre un tableau, sa taille (maximum et effective) ainsi que l'élément à ajouter, et délivre la taille effective du tableau.

```
Fonction Entier ajoutElentTabEntiers(S Entier
tab:tableau[taillemax], Entier taille, Entier taillemax,  Entier elem)
//Déclaration locale :
 Entier nouvelleTaille
Début
 nouvelleTaille←taille
 Si (taille < taillemax) Alors
  tab[taille]←elem
  nouvelleTaille←nouvelleTaille+1
 Finsi
 Retourner (nouvelleTaille)
Fin
```

- Tableau tab avant l'appel de la fonction ajoutElentTabEntiers: taillemax=7, taille=6 :

10	14	5	20	16	1	

- Tableau tab après l'appel de la fonction ajoutElentTabEntiers (tab, 6, 21): taille=7

10	14	5	20	16	1	21

6.3.5) Recherche d'un élément dans un tableau

Dans l'approche itérative, il s'agit de comparer chaque élément du tableau à l'élément de référence, sans dépasser le dernier élément. Dès que l'élément cherché est trouvé, il est inutile de continuer, et cette situation provoque l'arrêt de l'itération.

```
Fonction Booléen RechercheElemTabEntiers (Entier tab:tableau
[taillemax], Entier taille, Entier elem)
Déclaration locale :
Entier i
Début
 i←0
  Booléen trouve←Faux, finBoucle←(taille=0)
Tantque(Non finBoucle)
 Faire
  Si (tab[i]=elem) Alors
     Trouve←Vrai
     finBoucle←Vrai
   Sinon
   DébutSinon
     i←i+1
    Si(i>=taille) Alors
    finBoucle←Vrai
  Finsi
```

Finsinon
Finsi
Fintantque
Retourner(trouve)
FinFct

- Tableau tab avant l'appel de la fonction

10	14	5	20	16	1	21

- RechercheElemTabEntiers : taillemax=7, taille=7 :

- Résultat trouvé après l'appel de la fonction RechercheElemTabEntiers (tab, 7, 5) : Vrai

6.3.6) Recherche de la position d'un élément dans un tableau

Avant de rechercher l'indice, l'élément de référence est admis absent. Le résultat de la recherche est choisi par convention à -1, qui est une valeur d'indice *impossible*. On compare le résultat trouvé à tous les éléments du tableau, en faisant varier l'indice de 1 jusqu'à (*taille*). Si la comparaison révèle une égalité, il suffit de relever la valeur de l'indice, et d'indiquer que la recherche est terminée. Si un même élément est présent plusieurs fois, seule la position de sa première occurrence est délivrée.

Fonction Entier positionElemTabEntiers(**Entier** tab:tableau[taillemax], **Entier** taille, **Entier** elem)
Déclaration locale
Entier i←1, position←1, lafin←0
Début
TantQue(Non lafin)
Faire
Si (i>=taille) **Alors**

```
 Lafin←0
 Sinon
 Début
 Si (tab[i]=elem) Alors
  position←i
   Lafin←1
  Sinon
  i←i+1
 Finsi
FinTantQue
Retourner (position)
Fin
```

- Tableau tab avant l'appel de la fonction positionElemTabEntiers: taillemax=7, taille=7 :

10	14	5	20	16	1	21

- Résultat trouvé après l'appel de la fonction positionElemTabEntiers (tab, 7, 5) : 3

6.3.7) Suppression d'un élément dans un tableau

Le travail de suppression consiste à retirer du tableau un élément désigné par son indice. Il faut que cet élément fasse partie des éléments du tableau. La difficulté réside dans la manière de combler le vide laissé par la disparition de l'élément. Pour traiter ceci, il faut décaler vers la gauche tous les éléments situés à des indices supérieurs à celui supprimé, c'est-à-dire ranger ces éléments à un indice diminué de 1 par rapport à leur emplacement antérieur. On commence par traiter les éléments les plus proches de l'élément qui est supprimé.

```
Procédure supprElemTabEntiers(E/S Entier tab:tableau[], E/S
Entier taille, Entier position)
 Déclarations locales
  Entier i←position, nouvtaille←taille
 Début
  Si (position ≤ taille) Alors
   Tantque (i < taille)
    Faire
    tab[i]←tab[i+1]
    i←i+1
   FinTantQue
   nouvtaille←nouvtaille-1
   taille←nouvtaille
  Finsi
Fin
```

- Tableau tab avant l'appel de la procédure supprElemTabEntiers: taillemax=7, taille=7 :

10	14	5	20	16	1	21

- Tableau tab après l'appel de la procédure supprElemTabEntiers (tab, 7, 5) : taille devient égale à 6

10	14	5	20	1	21

6.3.8) Insertion d'un élément dans un tableau

L'insertion d'un élément n'est réalisable que si l'indice choisi pour cette insertion est inférieur ou égal à la taille effective du tableau. Il faut aussi que le tableau ne soit pas complètement plein pour pouvoir accueillir un nouvel élément. Une fois que ces conditions vérifiées, une place doit être réservée au nouvel élément. Cependant, il faut décaler vers la droite tous les éléments situés à des indices supérieurs à celui ajouté, c'est-à-dire

qu'il faut ranger les éléments à un indice augmenté de 1 par rapport à leur emplacement antérieur.

```
Fonction Entier insertElemTabEntiers(E/S Entier
tab:tableau[taillemax], E/S Entier taille, Entier position, Entier
element)
 Déclarations locales
 Entier dernier←taille, nouvtaille←taille
 Début
  Si ((taille≤taillemax) ET (position≤taille)) Alors
   TantQue (dernier≥position)
    Faire
     tab[dernier+1]←tab[dernier]
     dernier←dernier-1
    FinTantQue
    tab[position]←element
    nouvtaille←nouvtaille+1
  FinSi
  Retourner (nouvtaille)
FinFct
```

- Tableau tab avant l'appel de la procédure **insertElemTabEntiers:** taillemax=8, taille=7 :

10	14	5	20	16	1	21	

- Tableau tab après l'appel de la procédure insertElemTabEntiers (tab, 7, 4, 50): taille devient égale à 8

10	14	5	50	20	16	1	21

6.3.9) Inversion de l'ordre des éléments d'un tableau

Le principe d'inversion des éléments d'un tableau procède de la manière suivante : si le tableau contient plus d'un élément, le premier et le dernier élément sont permutés, puis on effectue l'inversion sur le sous-tableau privé de ses éléments extrêmes. Ceci est réalisé à l'aide de 2 variables qui désignent le premier et le dernier élément et qui sont mises à jour à chaque itération, jusqu'à ce que qu'elles se croisent.

```
Procédure permuterEntiers(E/S Entier a, E/S Entier b)
 Déclarations locales
  Entier tmp
 Début
  tmp←a
  a←b
  b←tmp
 FinProc

Procédure inversElemTabEntiers(E/S Entier tab:tableau[], Entier taille)
 Entier debut←1, dernier←taille
Début
 TantQue (dernier ≥ debut)
 Faire
  permuterEntiers(tab[debut], tab[dernier])
  debut←debut+1
  dernier←dernier-1
 FinTantQue
FinProc
```

- Tableau tab avant l'appel de la procédure inversElemTabEntiers : taillemax=8, taille=8 :

10	14	5	50	20	16	1	21

- Tableau tab après l'appel de la procédure inversElemTabEntiers (tab, 8) :

21	1	16	20	50	5	14	10

6.4) Applications sur les ensembles

Les ensembles sont une collection d'informations sans redondance. Chaque information n'est donc présente qu'une fois au plus dans un ensemble.

- **Application 1 :** *Ecrire une fonction qui permet de vérifier si les éléments d'un tableau d'entiers peuvent constituer un ensemble :*

- **Solution :**

```
Fonction Booléen verifierEnsemble(Entier tab:tableau[],
Entier n)
 Déclarations locales
 Entier i, j
 Booléen res←vrai
 Début
 Pour i de 1 à n-1
  Faire
   Pour j de i+1 à n
   Faire
    Si (tab[i]=tab[j]) Alors
     res←faux
    Finsi
    Finpour
```

```
 Finpour
 Retourner (res)
FinFct
```

Application 2 : *Si un tableau contient des éléments présents plusieurs fois, on peut le réduire à un ensemble en ne gardant qu'un exemplaire de chaque valeur.* Écrire *la fonction constituerEnsemble permettant de supprimer les doublons à partir d'un tableau d'entiers.*
Indications *: Pour tous les éléments du tableau, on recherche dans la partie restante les éléments identiques. S'il y en a, l'élément courant n'est pas conservé.*

Solution :

```
/* supprimer les "doublons" */
 Fonction Entier constituerEnsemble(E/S Entier
tab:tableau[], Entier n)
  Déclaration locale
  Entier i, taille←1
 Début
  Pour i de 1 à n
   Faire
  Si (Non RechercheElemTabEntiers (tab[i+1], n-i, tab[i]))
Alors
    taille←ajoutElentTabEntiers(tab, taille, n, tab[i])
  Finsi
 FinPour
 Retourner (taille)
FinFct
```

Application 3 : *L'union de 2 ensembles produit un ensemble contenant les éléments des 2 ensembles de départ. Pour conserver l'unicité, seuls les éléments du deuxième ensemble qui ne sont pas déjà dans le premier doivent être ajoutés. Écrire la procédure unionEns permettant de réunir deux ensembles quelconques.*

Solution :

Procédure UnionEns (Entier tab1:tableau[], Entier n1, Entier tab2:tableau[], Entier n2, **S** Entier tab3:tableau[], Entier max3)
Entier i, taille←1
Début
Pour i de 1 à n1
Faire
taille←ajoutElentTabEntiers(tab3, taille, max3, tab1[i])
FinPour
Pour i de 1 à n2
Faire
Si (Non RechercheElemTabEntiers(tab1, n1, tab2[i]))
Alors
taille←ajoutElentTabEntiers(tab3, taille, max3, tab2[i])
Finsi
FinPour
FinProc

- Les tableaux tab1 et tab2 de tailles respectives n1=5 et n2=6 :

Tab1

14	20	16	1	21

Tab2

10	5	50	20	16	1

- Tableau tab3 après l'appel de la fonction UnionEns (max3=11) :

Tab3

14	20	16	1	21	10	5	50

Application 4 : L'intersection *de deux ensembles réunit les éléments qui appartiennent à la fois au premier et au second ensemble. Écrire la procédure qui permet de faire l'intersection de deux ensembles différents.*

Solution :

```
Procédure IntersectionEns (Entier tab1:tableau [], Entier
n1, Entier tab2:tableau [], Entier n2, S Entier tab3 : ta-
bleau [], Entier max3)
  Déclarations locales
Entier i, taille←1
Début
Pour i de 1 à n1
Faire
Si (RechercheElemTabEntiers(tab2, n2, tab1[i])) Alors
Taille←ajoutElentTabEntiers(tab3,taille,max3,tab1[i])
FinSi
FinPour
Retourner (taille)
FinProc
```

Application 5 : *La différence consiste à retirer du premier ensemble les éléments qui n'appartiennent pas au second. En pratique, les éléments contenus dans l'ensemble de différence sont tous les éléments du premier sauf ceux présents dans le second. Écrire une procédure qui définit la fonction differenceEns.*

Solution :

```
Procedure differenceEns (Entier tab1:tableau[], Entier
n1, Entier tab2:tableau[], Entier n2, E/S Entier
tab3:tableau[], Entier max3)
  Entier i, taille←1
   Début
   Pour i de 1 à n1
   Faire
    Si (Non RechercheElemTabEntiers(tab2, n2, tab1[i]))
Alors
     taille←ajoutElentTabEntiers(tab3,taille,max3,tab1[i])
    FinSi
   FinPour
FinProc
```

- Les tableaux tab1 et tab2 de tailles respectives n1=5 et n2=6 :

Tab1

14	20	16	1	21

Tab2

10	5	50	20	16	1

- Tableau tab3 après l'appel de la fonction differenceEns

Tab3 :

14	21

Application 6 : *Écrire l'algorithme principal permettant de faire les opérations suivantes :*

- *Saisir 3 ensembles de 10 valeurs dans 3 tableaux d'entiers tab1, tab2 et tab3*
- *Tester si tab1, tab2 et tab3 sont des ensembles ou non*
- *Trouver l'union entre tab1 et tab2*
- *Trouver l'intersection entre tab1 et tab2*
- *Trouver la différence entre tab1 et tab2*

Solution :

```
Procédure afficheEnsemble(Entier tab:tableau, Entier n)
 Entier i
 Début
 Pour i de 1 à n
  Faire
  Ecrire(tab[i])
 FinPour
FinProc
//Déclarations
D.E: Entier tab1:tableau[10], tab2:tableau[10],
tab3:tableau[10]
D.S : Entier n1, n2, n3
Début
Ecrire("entrez 10 valeurs dans tab1 RC")//RC c'est-à-dire
Retour chariot
 Pour i de 1 à 10
  Faire
```

```
Lire(tab1[i])
FinPour
Ecrire("entrez 10 valeurs dans tab2 RC")
Pour i de 1 à 10
 Faire
 Lire(tab2[i])
FinPour
Ecrire("entrez 10 valeurs dans tab3 RC")
Pour i de 1 à 10
Faire
 Lire(tab3[i])
FinPour
Si(verifierEnsemble(tab1, 10)) Alors
 Ecrire("tab1 est un ensemble RC")
 Sinon Ecrire("tab1 n'est pas un ensemble RC")
 FinSi
Si (verifierEnsemble(tab2, 10)) Alors
 Ecrire("tab2 est un ensemble RC")
 Sinon Ecrire("tab2 n'est pas un ensemble RC")
FinSi
Si (verifierEnsemble(tab3, 10)) Alors
 Ecrire("tab3 est un ensemble RC")
 Sinon Ecrire("tab3 n'est pas un ensemble RC")
FinSi
n1←constituerEnsemble(tab1, 10)
n2←constituerEnsemble(tab2, 10)
n3←constituerEnsemble(tab3, 10)
Ecrire("Ensemble tab1 RC"); afficheEnsemble(tab1, n1)
Ecrire("Ensemble tab2 RC"); afficheEnsemble(tab2, n2)
Ecrire("Ensemble tab3 RC"); afficheEnsemble(tab3, n3)
n3←unionEns(tab1, n1, tab2, n2, tab3, 10)
Ecrire("Ensemble tab1 union tab2")
afficheEnsemble(tab3, n3)
n3←intersectionEns(tab1, n1, tab2, n2, tab3, n)
Ecrire("Ensemble tab1 inter tab2 RC")
afficheEnsemble(tab3, n3)
 n3←differenceEns(tab1, n1, tab2, n2, tab3, n)
Ecrire("Ensemble tab1 moins tab2 RC")
```

```
  afficheEnsemble(tab3, n3);
Fin
```

▸▸ Résultat d'exécution de l'algorithme principal :

```
entrez 10 valeurs dans tab1    1 2 3 4 5 6 1 2 7 8
entrez 10 valeurs dans tab2    1 1 1 2 5 6 7 8 9 0
entrez 10 valeurs dans tab3    0 1 2 3 4 5 6 7 8 9
tab1 n'est pas un ensemble
tab2 n'est pas un ensemble
tab3 est un ensemble
Ensemble tab1  3 4 5 6 1 2 7 8
Ensemble tab2  1 2 5 6 7 8 9 0
Ensemble tab3  0 1 2 3 4 5 6 7 8 9
Ensemble tab1 union tab2  3 4 5 6 1 2 7 8 9 0
Ensemble tab1 inter tab2  5 6 1 2 7 8
Ensemble tab1 moins tab2  3 4
```

6.5) Les tableaux bidimensionnels

Un tableau à deux dimensions est une représentation qui correspond à la notion de matrice en mathématiques qui se ressemble à l'organisation des tables obtenues par interrogation des bases de données (2 indices pour représenter les lignes et les colonnes). Par exemple, les températures sur les douze mois des années 2004 à 2010 sont naturellement représentées par une structure de tableau à deux dimensions. Le premier type d'indice entier 2004...2010 désigne l'année (indice de ligne), et le deuxième type d'indice entier 1...12 indique les douze mois de l'année.

En mathématique, on désigne chaque élément d'une matrice par a_{ij} où i est le numéro de ligne et j celui de colonne. En programmation, il en va de même : si M est une matrice de taille m lignes et n colonnes, on la déclare à l'aide d'un tableau par :

➡ **Réel** M : tableau [m][n]

Nous présentons un exemple de déclaration d'une matrice (point) de dimension [2]x[6] :

17	2	9	11	10	50
10	5	50	20	16	1

Déclaration : Réel M : tableau [2][6]

Comme pour les tableaux mono-dimensionnels, la numérotation commence à partir de 1 pour les lignes et pour les colonnes, donc l'élément a_{ij} est obtenu par M[i][j], ou réciproquement, l'élément M[i][j] d'un tableau est celui situé à la *(i)*ème ligne et *(j)*ème colonne. L'adresse de l'élément M[i][j] est celle du premier + (*i**nombre de colonnes) + *j*.

- **Accès en lecture** : Ecrire (M [1][6]) *{la valeur contenue en ligne 1 colonne 6 est affichée à l'écran}*
- **Accès en écriture** :
  - **M**[2][4] ←36
  - **Lire**(M[2][4]) *{la valeur fournie est enregistrée en ligne 2, colonne 4}*

Remarque L'affectation entre deux tableaux, s'effectue que si les deux tableaux possédant les mêmes caractéristiques (même type de composants et mêmes types d'indices), de même pour les comparaisons d'égalité et d'inégalité entre deux tableaux.

Exercice : *Ecrire un algorithme permettant de remplir un tableau de dimension 2 :*

Solution

```
Constantes : Entier tailleMAX ← 100
Variables :
D.E : entier nbLignes, indL,indC,
            nbColonnes , nombres: tableau
[1..tailleMax][1..tailleMAX]
Début
 Répéter
```

```
    Ecrire ("Combien de lignes?")
    Lire (nbLignes)
    Ecrire ("Combien de Colonnes?")
     Lire (nbColonnes)
     {Refuser la saisie : la capacité du tableau est dépassée}
     Ecrire ("trop de valeurs à saisir")
    Jusqu'à ((nbLignes<tailleMAX) ET (nbCo-
lonnes<tailleMAX))
    Pour indL de 1 à nbLignes faire
    Pour indC de 1 à nbColonnes faire
     Ecrire ("Donner une valeur")
     {valeur à ranger dans la indème case du tableau}
      Lire (nombres[indL] [indC])
    Finpour
   Finpour
  Fin
```

Exercices du chapitre 6

Exercice 6.1

Écrire l'algorithme permettant de :

1) Lire la dimension n d'un tableau tab du type entier (dimension maximale : 100 éléments)
2) Remplir le tableau par des valeurs entrées au clavier.
3) Calculer et afficher la somme des éléments du tableau.

Exercice 6.2

Écrire un algorithme qui permet de déclarer et de remplir un tableau contenant les 6 voyelles de l'alphabet français.

Exercice 6.3

Écrire un algorithme qui permet de saisir les notes de n élèves d'une classe dans un devoir et les mémoriser dans un tableau de dimension n. Cet algorithme permet de rechercher et d'afficher :

- la note maximale,
- la note minimale,
- la moyenne des notes.

Exercice 6.4

Écrire un algorithme qui permet de rechercher le plus grand nombre paire d'un tableau d'entiers de taille n.

Exercice 6.5

Écrire un algorithme recherche la plus grande différence entre deux cases consécutives d'un tableau de taille n.

Exercice 6.6

Écrire un algorithme qui fait la multiplication de deux tableaux tab1 et tab2 de tailles respectives n1 et n2 et d'afficher le résultat de multiplication.

Principe : La multiplication s'effectue à travers le produit cartésien des éléments de tab1 et tab2 et le résultat retourné désigne la somme des multiplications réalisées.

Exercice 6.7

Étant donné un texte de taille t représenté dans un tableau défini sur l'intervalle [1...tmax]. On supposera dans les questions ci-dessous que le tableau considéré contient déjà le texte, sa saisie dans le tableau n'est donc pas à faire.
a) Ecrire un algorithme qui affiche un message indiquant si dans le texte donné, l'une des voyelles ('a', 'e', 'i', 'o', 'u', 'y') est absente et laquelle.
b) Ecrire un algorithme qui affiche un message indiquant si le texte donné est un palindrome (une fois les espaces supprimés, le texte se lit de gauche à droite comme de droite à gauche). On suppose que n est déjà initialisé et le texte déjà rempli.
Exemples de palindrome :

- esope reste ici et se repose
- tu l as trop ecrase cesar ce port salut

Exercice 6.8

Etant donné un tableau d'entiers t défini sur [1...n] avec n > 0 supposé initialisé, écrire un algorithme qui réalise le décalage circulaire à droite de T selon la spécification suivante : soient A et B les valeurs initiale et finale du tableau T. Le décalage est caractérisé par l'assertion : B[1] = A[n] et $\forall$ k $\in$[2...n], B[k]= A[k-1].

a) Écrire une première solution dans laquelle le problème est sérialisé en deux étapes : on crée tout d'abord un tableau intermédiaire z, puis on recopie z dans t.

b) Écrire une deuxième solution sans recourir à un tableau intermédiaire.

Exemple :

- avant décalage circulaire à droite (avec n= 6) T = [2, 3, 29, 54, -3, 12]
- après décalage circulaire à droite (avec n= 6) T = [12, 2, 3, 29, 54, -3]

Exercice 6.9

Un tableau de taille n contient les nombres 0, 1, 2 et 3 représentant respectivement les lettres A, T, G ou C d'une chaîne d'ADN.

1) Écrire un algorithme qui compte le nombre d'occurrences de la succession des lettres A, T, A (isoleucine)

Deux tableaux sont de taille n et contiennent les entiers 0, 1, 2 ou 3 (respectivement pour A, T, C ou G) représentent deux chaînes complémentaires de l'ADN.

2) Écrire un algorithme qui compte le nombre de bases non correctement appariées.

Exercice 6.10

Écrire un algorithme qui permet de :

1) Lire les dimensions l et c d'un tableau mat (matrice) à deux dimensions du type **entier** (dimensions maximales : 3 lignes et 3 colonnes).
2) Remplir le tableau par des valeurs entrées au clavier, puis afficher la somme de tous ses éléments.

Exercice 6.11

Écrire un algorithme qui permet de :

1) lire les dimensions *l* et *c* d'un tableau mat à deux dimensions du type **entier** (dimensions maximales : 3 lignes et 3 colonnes).
2) Remplir le tableau par des valeurs entrées au clavier et d'afficher la somme de chaque ligne et de chaque colonne en n'utilisant qu'une variable d'aide pour la somme.

Indication : Pour la matrice mat suivant :

1	2	3
5	6	7
4	5	3

Somme ligne 0=6
Somme ligne 1=18
Somme ligne 2=12
Somme colonne 0=10
Somme colonne 1=13

Somme colonne 2=13

Exercice 6.12

Écrire un algorithme qui permet de faire l'addition de deux matrices a et b de mêmes dimensions n et m dans une matrice c.
Indication :

a

1	2	3
5	6	7
4	5	3

+

b

1	2	3
5	6	7
4	5	3

=

c

2	4	6
10	12	14
8	10	6

La matrice b est ajoutée à a. Le résultat de l'addition sera mémorisé dans une troisième matrice c qui sera ensuite affichée.

Exercice 6.13

Écrire un algorithme qui construit le triangle de PASCAL de degré n et le mémorise dans une matrice carrée p de dimension n+1.
Exemple: Triangle de Pascal de degré 6:

1						
1	1					
1	2	1				
1	3	3	1			
1	4	6	4	1		
1	5	10	10	5	1	
1	6	15	20	15	6	1

Indications : Calculer et afficher seulement les valeurs jusqu'à la diagonale principale (incluse). Limiter le degré à entrer par l'utilisateur à 13. Construire le triangle ligne par ligne :

- Initialiser le premier élément et l'élément de la diagonale à 1.
- Calculer les valeurs entre les éléments initialisés de gauche à droite en utilisant la relation : $Pi,j = P_{i-1,j} + P_{i-1,j-1}$

Exercice 6.14

Écrivez les sous-algorithmes suivants :

1) **set_0** qui initialise une matrice par les valeurs réelles nulles ;
2) **set_Id** qui initialise une matrice à la matrice Identité. La fonction set_Id retournera :
   - *Faux* si la matrice passée en paramètre n'est pas carrée, et
   - *Vrai* si l'opération s'est bien passée ;
3) **init** permettant d'initialiser une matrice.

Il faut écrire deux procédures différentes pour chaque taille de matrice que vous voudrez initialiser :

- Procédure init (Entier nl, Entier nc, Réel c1, Réel c2, Réel c3, Réel c4, S Réel mat:tableau [][]) permettant d'initialiser une matrice 2x2;
- Procédure init (Entier nl, Entier nc, Réel c1, Réel c2, Réel c3, Réel c4, Réel c5, Réel c6, S Réel mat:tableau [][]) permettant d'initialiser une matrice 2x3 ou 3x2.

Ces sous_algorithmes devront donc tester la valeur de nl et nc pour gérer les cas, et mettre les paramètres c1, c2... au bon endroit.

4) **add** permettant d'additionner deux matrices.

Corrigé des exercices du chapitre 6

Exercice 6.1 (Solution)

D.E : **Entier** tab : tableau [100], n
D.I : Entier i
D.S : Entier Som
Début
Répéter
Ecrire (''Donner la taille du tableau'')
Lire(n)
Jusqu'à (n>0 **ET** n<100)
som←0
Pour i de 1 à n **Faire**
Ecrire (''Donner l'entier d'ordre **Entier**'', i)
Lire (tab[i])
som←som+tab[i]
FinPour
Ecrire (''La somme des éléments du tableau est un **Entier** :'', som)
Fin

Exercice 6.2 (Solution)

D.E : **Caractère** voy : tableau [6], c
D.I : Entier i
Début
i←1
Répéter

```
   Ecrire ("Donner un caractère")
    Lire(c)
   Si (c='a' OU c='i' OU c='u' OU c='e' OU c='o' OU  c='y')
Alors
    voy[i]=c;
    i←i+1
   Finsi
  Jusqu'à (i=6)
Fin
```

Exercice 6.3 (Solution)

```
D.E : Entier n ; Réel notes : tableau [n]
D.I : Entier i
D.S : Réel  notemax, notemin, moy
Début
Répéter
   Ecrire ("Donner la taille du tableau")
   Lire(n)
Jusqu'à (n>0)
Pour i de 1 à n Faire
   Ecrire ("Donner la note d'ordre Entier", i)
   Lire (notes[i])
FinPour
moy←0, notemax←notes[0], notemin←notes[0] ;
Pour i de 1 à n Faire
   moy←moy+(notes[i] div n)
   Si(notemax<notes[i]) Alors
    notemax←notes[i]
    Finsi
    Si(notemin>notes[i]) Alors
```

```
    notemin←notes[i]
    Finsi
 FinPour
Ecrire ("La moyenne des notes est =Réel :", moy)
Ecrire ("La note max est=Réel :", notemax)
Ecrire ("La note min est=Réel :", notemin)
Fin
```

Exercice 6.4 (Solution)

```
D.E : Entier n, tab : tableau [n]
D.I : Entier i, j
D.S : Entier pnbpaire
Début
i←1
   TantQue(tab[i]%2<>0 Et  i<>n)
   pnbpaire←tab[i]
   i←i+1
   FinTantque
  Si (i=n) Alors  Ecrire("Il n' existe aucun nombre paire")
    Sinon
   Pour j de i+1 à n Faire
   Si(tab[j]%2=0 ET tab[j]>pnbpaire) Alors
    pnbpaire←tab[j]
   Finsi
  FinPour
Ecrire ("Le plus grand élément pair du tableau est =Entier :",
pnbpaire)
Finsi
Fin
```

Exercice 6.5 (Solution)

```
D.E : Entier n, tab : tableau [n]
D.I : Entier i
D.S : Entier granddiff, diff
Début
   Si (tab[1]-tab[2]>0)  Alors
    granddiff←tab[1]-tab[2]
    Sinon
    granddiff←tab[2]-tab[1]
   Finsi
   Pour i de 2 à n-1 Faire
    Si(tab[i]-tab[i+1]>0) Alors
     diff←tab[i]-tab[i+1]
     Sinon
     diff←tab[i+1]-tab[i]
    Finsi
    Si (diff>granddiff) Alors
     granddiff←diff
    Finsi
  FinPour
Ecrire ("La plus grande différence entre deux cases consécutives est =Entier :", granddiff)
Fin
```

Exercice 6.6 (Solution)

```
D.E : Entier n1, n2, tab1 : tableau [n1], tab2 : tableau [n2]
D.I : Entier i, j
D.S : Entier prodtab
Début
```

```
   prodtab←0
   Pour i de 1 à n1 Faire
    Pour j de 1 à n2 Faire
      prodtab← prodtab+tab1[i]*tab2[j]
    FinPour
   FinPour
Ecrire ("La valeur du produit de tab1 par tab2 est un Entier:",
prodtab)
Fin
```

Exercice 6.7 (Solution)

a)

```
D.E : Entier t, tmax, n
      Caractère voy : tableau []={'a', 'e', 'i', 'o', 'u', 'y'}, texte: ta-
bleau [tmax]
D.I : Entier i, j
      Booléen exist, insert
D.S : Caractère existvoy : tableau [n]
Début
 i←1
Pour i de 1 à t
Faire
j←1
insert←Faux
Pour k de 1 à n
Faire
  Si(existvoy[k]=texte[i]) Alors
   insert←Vrai
  Finsi
 FinPour
exist←Faux
```

```
Si (texte[i]='a' OU texte[i]='i' OU texte[i]='u' OU texte[i]='e'
OU texte[i]='o' OU  texte[i]='y') Alors
exist←Vrai
Finsi
 Si (exist=Vrai ET insert=Vrai) Alors
  existvoy[j]←text[i]
  j←j+1
 Finsi
FinPour
//Vérifier la voyelle absente
Pour i de 1 à 6
 Faire
   exist←Faux
    Pour k de 1 à j Faire
      Si (existvoy[k]= voy [i]) Alors
       exist←Vrai
      Finsi
    FinPour
    Si (exist=Faux) Alors
     Ecrire ("La voyelle caractère n'existe pas dans le texte",
voy [i]) ;
   Finsi
FinPour
Fin
```

b)

```
Procédure supprimerEspaces(Entier n, E/S  Caractère texte :
tableau [n], S Entier t)
Début
 t←n
Pour i d 1 à n
 Faire
```

```
  Si (texte[i]='' '') Alors
   texte[i]=texte[i+1]
   t←t-1
  Finsi
Finpour
FinProc

D.E : Entier n, t, Caractère texte : tableau [n]
D.I : Entier i, j
D.S : Entier prodtab
Début
  //Le tableau est supposé rempli et n connu
    prodtab←0
   supprimerEspaces(n, texte, t)
    i←t
    j←1
   TantQue (i<>j ET texte[i]=texte[j])
   Faire
    i←i-1
    j←j+1
    FinTantQue
     Si(i=j) Alors
    Ecrire ("Le texte est palindrôme")
    Sinon
    Ecrire ("Le texte est non palindrôme")
   Finsi
Fin
```

Exercice 6.8 (Solution)

a)

```
D.E : Entier n, t : tableau [n]
```

```
D.I : Entier i, z : tableau [n], aux
Début
   z[1]←t[n]
   Pour i de 2 à n Faire
     z[i]←t[i-1]
   FinPour
   Pour i de 1 à n Faire
     t[i]←z[i]
   FinPour
Fin
```

b)

```
D.E : Entier n, t : tableau [n]
D.I : Entier i, aux
Début
   aux←t[n]
   Pour i de 2 à n Faire
     t[i]←t[i-1]
   FinPour
   t[1]←aux
Fin
```

Exercice 6.9 (Solution)

1)

```
D.E : Entier n, adn : tableau [n]
D.I : Entier i
D.S : Entier nbocc
Début
   nbocc←0
   Pour i de 1 à n-2 Faire
     Si (adn[i]=0 ET adn[i+1]=1 ET adn[i+2]=0) Alors
        nbocc←nbocc+1
```

```
    Finsi
  FinPour
Ecrire ("Le nombre d'occurrences d'isoleucine est un entier",
nbocc)
Fin
```

2)

```
D.E : Entier n, adn1 : tableau [n], adn2 : tableau [n]
D.I : Entier i
D.S : Entier nbnapp
Début
   nbnapp←0
   Pour i de 1 à n Faire
     Si (adn1[i]<> adn2[i])) Alors
        nbnapp ← nbnapp +1
     Finsi
  FinPour
Ecrire ("Le nombre de bases non correctement appariées est =
Entier", nbnapp)
Fin
```

Exercice 6.10 (Solution)

```
D.E : Entier l, c, mat : tableau [3][3]
D.I : Entier i, j
D.S : Entier som
Début
Répéter
  Ecrire ("Donner les dimensions de lignes et de colonnes :") ;
  Lire (l,c)
Jusqu'à (l≤3 et c≤3)
  som←0
```

```
Pour i de 1 à l Faire
  Pour j de 1 à c Faire
    Ecrire (" Donner l'élément de la case (entier, entier)",i, j)
    Lire (mat[i][j])
   som←som+mat[i][j]
   FinPour
FinPour
Ecrire ("La somme des éléments de la matrice est un entier",
som)
Fin
```

Exercice 6.11 (Solution)

```
D.E : Entier l, c, mat : tableau [3][3]
D.I : Entier i, j
D.S : somL, somC
Début
Répéter
   Ecrire ("Donner les dimensions de lignes et de colonnes :")
   Lire (l,c)
  Jusqu'à (l≤3 ET c≤3)
  Pour i de 1 à l Faire
    Pour j de 1 à c Faire
    Ecrire (" Donner l'élément de la case (Entier, Entier)", i, j)
    Lire (mat[i][j])
FinPour
Finour
//Calcul et affichage de la somme des lignes
  Pour i de 1 à l Faire
    somL←0
    Pour j de 1 à c Faire
```

```
   somL←somL+mat[i][j]
  FinPour
Ecrire ("La somme des éléments de la ligne numéro Entier est =
Entier", i, somL)
FinPour
//Calcul et affichage de la somme des colonnes
Pour i de 1 à c Faire
   somC←0
   Pour j de 1 à l Faire
     somC←somC+mat[j][i]
  FinPour
Ecrire ("La somme des éléments de la colonne numéro Entier
est = Entier", i, somC)
FinPour
Fin
```

Exercice 6.12 (Solution)

```
D.E : Entier l, c, a : tableau [3][3], b : tableau [3][3]
D.I : Entier i, j
D.S :  c : tableau [3][3]
Début
Répéter
   Ecrire ("Donner les dimensions de lignes et de colonnes :")
   Lire (l,c)
  Jusqu'à (l≤3 ET c≤3)

Pour i de 1 à 3 Faire
 Pour j de 1 à 3 Faire
   Ecrire (" Donner l'élément de la case (Entier, Entier) de la
matrice a", i, j)
```

```
   Lire (a[i][j])
 FinPour
FinPour
Pour i de 1 à 3 Faire
 Pour j de 1 à 3 Faire
   Ecrire (" Donner l'élément de la case (Entier, Entier) de la
matrice b", i, j)
   Lire (b[i][j])
 FinPour
FinPour
Pour i de 1 à 3 Faire
 Pour j de 1 à 3 Faire
   c[i][j]←a[i][j]+b[i][j]
 FinPour
FinPour
Pour i de 1 à 3 Faire
 Pour j de 1 à 3 Faire
  Ecrire ("Entier ", c[i][j])
 FinPour
Ecrire (RC)
FinPour
Fin
```

Exercice 6.13 (Solution)

```
D.E : Entier deg
D.I : Entier i, j
D.S : Entier p : tableau [deg+1][deg+1]
Début
Répéter
```

```
   Ecrire ("Donner le degré du triangle :")
   Lire (deg)
  Jusqu'à (deg≤13)
 //Initialisation du triangle
Pour i de 1 à deg+1 Faire
 Pour j de 1 à deg+1 Faire
  Si (j=1 OU j=i) Alors
   p[i][j]←1
   Sinon
   p[i][j]←0
  Finsi
FinPour
FinPour
Pour i de 3 à deg+1 Faire
 Pour j de 2 à deg+1 Faire
     p[i][j]←p[i-1][j]+ p[i-1][j-1]
FinPour
FinPour
Fin
```

Exercice 6.14 (Solution)

1)

```
Procédure set_0(Entier nl, Entier nc, E/S  Réel mat:tableau [][])
Début
Pour  i = 1 à nl Faire
Pour  j= 1 à nc Faire
   mat[i][j] ← 0
FinPour
FinPour
FinProc
```

2)

```
Booléen fonction set_Id (Entier nl, Entier nc, E/S Réel mat :
tableau)
Début
Si (nl <> nc) Alors
 Retourner (Faux)
 Sinon
 Débutsinon
 set_0(nl, nc, mat)
 Pour i de 1 à nl Faire
  mat[i][i]←1
 FinPour
  Retourner (Vrai);
  FinSinon
Finsi
FinFct
```

3)

a)

```
Procédure init (Entier nl, Entier nc, Réel c1, Réel c2, Réel c3,
Réel c4, S Réel mat:tableau [][])
Début
 Si  (nl = 2 ET nc = 2) Alors
   mat[1][1] ← c1;
   mat[1][2] ← c2;
  mat[2][1] ← c3;
  mat[2][2] ←c4;
 Finsi
FinProc
```

b)

```
Procédure init (Entier nl, Entier nc, Réel c1, Réel c2, Réel c3,
Réel c4, Réel c5, Réel c6, S Réel mat:tableau [][])
Début
```

```
Si  (nl = 3 ET nc =2) Alors
  mat[1][1] = c1
  mat[1][2] = c2
 mat[2][1] = c3
 mat[2][2] = c4
 mat[3][1] = c5
 mat[3][2] = c6
Sinon
Si (nl = 2 ET nc = 3) Alors
  mat[1][1] = c1
  mat[1][2] = c2
 mat[1][3] = c3
 mat[2][1] = c4
 mat[2][2] = c5
 mat[2][3] = c6
 Finsi
Finsi
FinProc
```

4)

```
Procédure add(Entier nl, Entier nc, Réel a:tableau [][], Réel
b:tableau [][], S Réel c:tableau [][])
Début
Pour i de 1 à nl
  Faire
 Pour j de 1 à nc
  Faire
  c[i][j] = a[i][j]+b[i][j]
 FinPour
FinPour
FinProc
```

Chapitre 7

Les méthodes de tri et de recherche dans un tableau

7.1) Le tri d'un tableau 257

7.1.1) Le tri naïf 258

7.1.2) Le tri par insertion 259

7.1.3) Le tri par sélection 265

7.1.4) Le tri à bulle (bubble sort) 271

7.1.5) Le tri par fusion 276

7.2) Les méthodes de recherches dans un tableau 280

7.2.1) Recherche séquentielle 281

7.2.2) Recherche dichotomique 284

CHAPITRE 7

Les méthodes de tri et de recherche dans un tableau

Résumé : **Le tri d'un tableau accélère de façon significative l'accès aux informations. Nous allons présenter, dans les paragraphes qui suivent, plusieurs manières de trier des tableaux d'entiers à travers des algorithmes et des sous-algorithmes. De même, la recherche d'informations à partir d'un tableau nécessite des algorithmes optimisés pour accélérer de façon acceptable la recherche d'informations. Nous présentons dans ce chapitre, les types d'algorithmes de tri et de recherches d'informations les plus connus.**

7.1) Le tri d'un tableau

Le tri est une opération qui consiste à répartir ou organiser une collection d'objets selon un ordre déterminé. Dans le domaine de l'informatique, il existe plusieurs méthodes de tri (algorithmes). Dans ce chapitre nous allons découvrir diverses méthodes de tri, comme expliquée dans les sections suivantes.

7.1.1) Le tri naïf

Principe : Une manière intuitive de trier par ordre croissant une collection d'informations (tableau) peut être décrite comme suit : Si le tableau contenant ces informations est vide, on peut le considérer comme trié. Sinon, on cherche la position de son plus petit élément, afin de le permuter avec le premier élément. À l'issue de cette étape, le premier élément est le plus petit, il ne reste plus qu'à trier le sous-tableau qui commence à partir de l'élément suivant.

Étapes : tri d'un tableau : version naïve

1 - recherche de la position du plus petit élément

2 - on échange cet élément avec celui au début du tableau

3 - on recommence, sans toucher au début qui est déjà trié

// Dans cette étape, le premier élément est le plus petit, il ne reste plus qu'à trier

D.E : Entier n, tab : **tableau**[n] // tableau non trié

D.S : Entier tab : **tableau**[n] trié

D.I : Entier i, p, e1, e2

Déclaration des Sous-algorithmes :

Procédure posPlusPetit(**E/S Entier** tab : **tableau**[], **Entier** n) début...Fin

Procédure permuterEntier(E/S Entier e1, Entier E/S e2)

Début

Ecire("Donner la taille du tableau tab") ;

Lire (n) ;

Pour i **de** 1 **à** n **Faire**

P←posPlusPetit(tab[i], n-i)

permuterEntiers(tab[p+i], tab[i])

FinPour

Fin

⏩ Définition du sous-algorithme posPlusPetit

```
Procédure posPlusPetit(E/S Entier tab:tableau[], Entier n)
   Déclaration locale :
   Entier i,pos←-1,min
Début
        Si (n>0) Alors
         min ← tab[1]
         pos ← 1
        finsi
        Pour i de 1 à n Faire
         Si (min > tab[i]) Alors
          min ← tab[i]
           pos ← i
         Finsi
        Finpour
       Retourner (pos)
FinProc
Procédure permuterEntier(E/S Entier e1, E/S Entier e2)
    Entier vaux
    Début
    vaux←e1
    e1←e2
    e2←vaux
FinProc
```

7.1.2) Le tri par insertion

Activité : Écrire un algorithme qui permet de saisir un tableau **T** de **n** entiers, puis trier en ordre croissant ce tableau en utilisant la **méthode de tri par insertion** et afficher le résultat.

Principe :

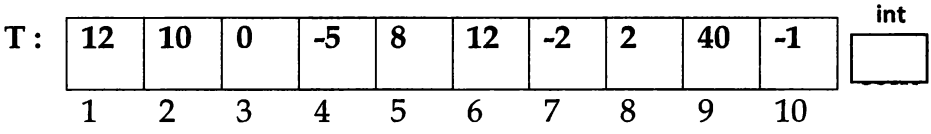
Cette méthode de tri consiste à :
Considérer que les **i-1** premiers éléments du tableau **T** sont triés et insérer l'élément **N°i** dans sa position parmi les **i-1** déjà triés.

Répéter cette action jusqu'à l'arrivé au dernier élément du tableau **T**.

Indication : L'insertion se traduit par la sauvegarde de l'élément **N° i** dans une variable intermédiaire **(int)**, puis le décalage vers une case à droite des éléments **i-1, i-2, ...** jusqu'à avoir un élément inférieur à "**int**" et finalement l'affectation du contenu de "**int**" dans l'élément libre.

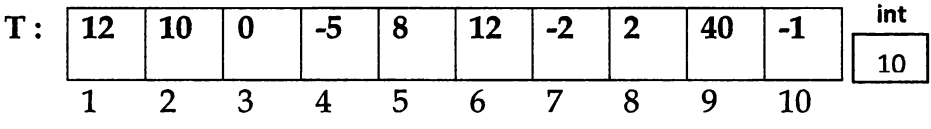
Exemple :

Soit un tableau T contenant les dix éléments suivants :

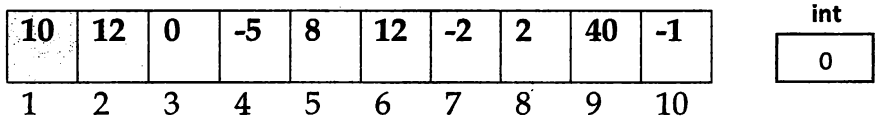


La recherche commence par l'élément N°2 puisque si le tableau contient un seul élément, il est considéré trié.

➡ **Etape1 :** Recherche de la position d'insertion de T[2]



On obtient :



➡ **Etape2 :** Recherche de la position d'insertion de T[3]

T :	10	12	0	-5	8	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

On obtient :

T :	0	10	12	-5	8	12	-2	2	40	-1
		2	3	4	5	6	7	8	9	10

➡ **Etape3 :** Recherche de la position d'insertion de T[4]

T :	0	10	12	-5	8	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

int
-5

On obtient :

T :	-5	0	10	12	8	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

➡ **Etape4 :** Recherche de la position d'insertion de T[5]

T :	-5	0	10	12	8	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

int
8

On obtient :

T :	-5	0	8	10	12	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

➡ **Etape5 :** Recherche de la position d'insertion de T[6]

T :	-5	0	8	10	12	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

On obtient :

T :	-5	0	8	10	12	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

int
12

➡ **Etape6 :** Recherche de la position d'insertion de T[7]

T :	-5	0	8	10	12	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

int
-2

On obtient :

T :	-5	-2	0	8	10	12	12	2	40	-1

➡ **Etape 7** : Recherche de la position d'insertion de T[8]

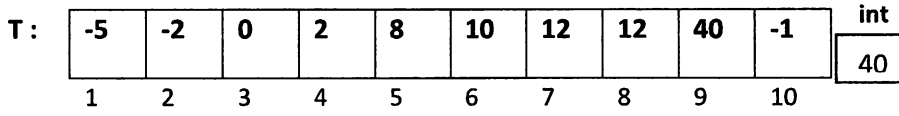
T :	-5	-2	0	8	10	12	12	2	40	-1
	1	2	3	4	5	6	7	8	9	10

int
2

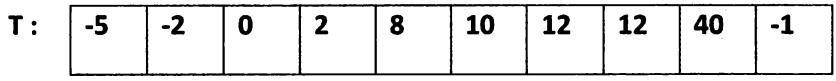
On obtient :

T :	-5	-2	0	2	8	10	12	12	40	-1

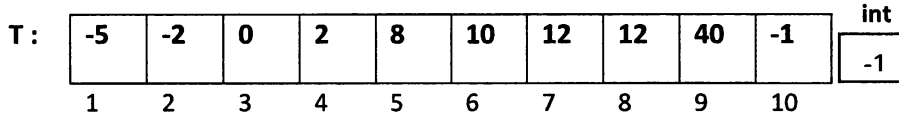
➡ **Etape 8** : Recherche de la position d'insertion de T[9]



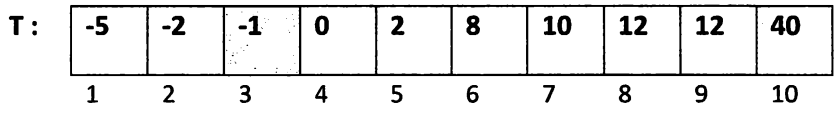
On obtient :



➡ **Etape 9** : Recherche de la position d'insertion de T[10]



On obtient :



Spécifications de l'algorithme de tri par insertion :

- **Résultat :** Afficher le tableau T trié en utilisant une procédure **Affiche**
- **Traitements :** Il faut trier le tableau T, en utilisant une procédure **Tri**
- **Données :** Il faut remplir le tableau T et saisir la taille n, en utilisant la procédure **Saisie**

Algorithme principal
Début
Saisie (n, T)
Tri (n, T)
Affiche (n, T)
Fin

➡ **Spécification de la procédure Tri :**

✓ **Résultat de sortie :** Trier le tableau **Tf**

✓ **Traitements :** Il s'agit d'un traitement répétitif pour tous les éléments du tableau, sauf le premier, d'où utilisation de la boucle Pour... Faire...,

✓ Pour chaque élément nous allons exécuter ces actions :

- **Action1 :** Chercher la position correcte de l'élément en question
- **Action2 :** Le ranger dans une variable intermédiaire (**int**)
- **Action3 :** Décaler d'une position les éléments supérieurs a cet élément
- **Action4 :** Affecter au dernier élément décalé la valeur de la variable **int**

Les paramètres formels pour cette procédure sont le tableau **tf** et sa taille **nf**.

```
Procédure Tri (Entier nf, E/S Entier tf : tableau [nf] )
Entier pos, int
Début
Pour i de 2 à n Faire
 pos ← 1
Tant que (pos < i) Faire
   Si (tf[pos] > tf[i]) Alors
       int ← tf[i]
       Pour j de i à pos +1 Faire
       tf[j] ← tf[j-1]
      FinPour
tf[pos] ←int
Finsi
pos ←pos +1
FinTantQue
Fin Pour
```

FinProc

7.1.3) Le tri par sélection

Principe : **Cette méthode de tri consiste à :**

Se pointer à la 1ère case du tableau T et de parcourir la totalité du tableau pour **repérer l'indice de la première position du minimum,**

1. **Comparer** ce minimum avec **T [1]**. S'ils sont différents, on **les permute** ;
2. **Le sous tableau** de T allant de **2 à n** est à priori non trié, on applique l'étape **1** et **2** et ainsi de-suite jusqu'à l'avant-dernier élément **(n-1)**.

Exemple : Soit un tableau T contenant les dix éléments suivants :

T :	12	10	0	-5	8	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

➡ **Etape1 :** Parcourir la totalité du tableau pour repérer le minimum (indice de la première position du minimum) et le comparer avec T [1]

T :	12	10	0	-5	8	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

Remarque : Indice du minimum=4 et T[1]<>T[4] alors il faut permuter

On obtient :

T :	-5	10	0	12	8	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

Le sous tableau allant de 2 à n est à priori non trié, on applique l'étape **1** et **2** et ainsi de suite jusqu'à l'avant dernier élément **(n-1)**.

➡ **Etape2 :**

T :	-5	10	0	12	8	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

Remarque : Indice du minimum=7 et T[2]<>T[7] alors il faut permuter

On obtient :

T :	-5	-2	0	12	8	12	10	2	40	-1
	1	2	3	4	5	6	7	8	9	10

➡ **Etape3 :**

T :	-5	-2	0	12	8	12	10	2	40	-1
	1	2	3	4	5	6	7	8	9	10

Remarque : Indice du minimum=10 et T[3]<>T[10] alors il faut permuter

On obtient :

T :	-5	-2	-1	12	8	12	10	2	40	0
	1	2	3	4	5	6	7	8	9	10

➡ **Etape4 :**

T :	-5	-2	-1	12	8	12	10	2	40	0
	1	2	3	4	5	6	7	8	9	10

Remarque : Indice du minimum=10 et T[4]<>T[10] alors il faut permuter

On obtient :

T :	-5	-2	-1	0	8	12	10	2	40	12
	1	2	3	4	5	6	7	8	9	10

➡ **Etape5 :**

T :	-5	-2	-1	0	8	12	10	2	40	12
	1	2	3	4	**5**	6	7	8	9	10

Remarque : Indice du minimum=8 et T[5]<>T[8] alors il faut permuter

On obtient :

T :	-5	-2	-1	0	2	12	10	8	40	12
	1	2	3	4	5	6	7	8	9	10

➡ **Etape6 :**

T :	-5	-2	-1	0	2	12	10	8	40	12
	1	2	3	4	5	6	7	8	9	10

Remarque : Indice du minimum=8 et T[6]<>T[8] alors il faut permuter

On obtient :

T :	-5	-2	-1	0	2	8	10	12	40	12
	1	2	3	4	5	6	7	8	9	10

➡ **Etape7 :**

T :	-5	-2	-1	0	2	8	10	12	40	12
	1	2	3	4	5	6	7	8	9	10

Remarque : Indice du minimum=7 et T[7]=T[7] alors il ne faut pas permuter

On obtient :

T :	-5	-2	-1	0	2	8	10	12	40	12
	1	2	3	4	5	6	7	8	9	10

➡ **Etape8 :**

T :	-5	-2	-1	0	2	8	10	12	40	12
	1	2	3	4	5	6	7	8	9	10

Remarque : Indice du minimum=8 et T[8]=T[7] alors il ne faut pas permuter

On obtient :

T :	-5	-2	-1	0	2	8	10	12	40	12
	1	2	3	4	5	6	7	8	9	10

➡ **Etape9 :**

T :	-5	-2	-1	0	2	8	10	12	40	12
	1	2	3	4	5	6	7	8	9	10

Remarque : Indice du minimum=10 et T[9]<>T[10] alors il faut permuter

On obtient :

T :	-5	-2	-1	0	2	8	10	12	12	40
	1	2	3	4	5	6	7	8	9	10

Remarque :

➡ On est arrivé à l'élément numéro n-1, alors arrêt du traitement.

⏩ Nous n'avons pas besoin de traiter le dernier élément, puisque si les neuf premiers éléments sont triés alors automatiquement le dernier sera le plus grand et par conséquent il se trouve à la bonne position.

1) Spécification de l'algorithme principal :

⏩ **Résultat :** Afficher le tableau T trié en utilisant une procédure **Affiche**

⏩ **Traitements :** Il faut trier le tableau T, en utilisant une procédure **Tri**

- **Données :** Il faut remplir le tableau T et saisir la taille n, en utilisant la procédure "**Saisie**".

Algorithme principal
Début
Saisie (n, T)
Tri (n, T)
Affiche (n, T)
Fin

2) Spécification de la procédure Saisie

- **Résultat :** Saisir **nf** et remplir **tf**
- **Traitements :** Le remplissage d'un tableau est une action répétitive, ou on connaît le nombre de répétition qui est égale à **nf**, d'où utilisation de la boucle Pour ... Faire... La saisie de l'entier **nf** doit être contrôlée pour ne pas saisir un entier négatif ou supérieur à 100. Cette procédure admet deux paramètres formels qui sont **nf** et **tf.**

Procédure Saisie **(E/S Entier** nf, E/S **Entier** tf :**tableau**[nf])
Début
Répéter
Ecrire ("Donner la taille du tableau : ")
Lire (nf)
Jusqu'à (nf>0 et nf<100)
Pour i **de** 1 **à** nf **Faire**
Ecrire ("Donner l'élément N° entier :", i)
Lire (tf[i])
Fin Pour
FinSaisie

3) Spécification de la procédure Tri :

- **Résultat :** Trier le tableau **tf**

- **Traitements :** Il s'agit d'un traitement répétitif jusqu'à l'avant dernier élément du tableau, d'où utilisation de la boucle Pour ... Faire ..., pour chaque élément nous allons exécuter deux actions :
- **Action1 :** Chercher la première apparition du minimum
- **Action2 :** Comparer l'indice du minimum à celui de l'élément en cours, s'ils sont différents, alors on applique la permutation à ces deux éléments.

Donc, on fera appel à une fonction intitulée "**premposmin**" qui retourne le premier indice du minimum et à une procédure intitulée "**permut**" permettant de permuter deux éléments. Les paramètres formels pour cette procédure sont le tableau "**tf**" et sa taille "**nf**".

```
Procédure Tri (Entier nf, E/S Entier tf :tableau[nf])
 Entier pmin
 Début
   Pour i de 1 à Nf-1 Faire
     pmin ← premposmin (nf, tf, i)
     Si (i <> pmin) Alors
     Permut (Tf[i], Tf[Pmin])
    Finsi
  Fin Pour
Fin
```

4) **Spécification de la fonction premposmin :**

- **Résultat :** Déterminer la première position du minimum dans un sous tableau
- **Traitements :** Il s'agit d'un traitement répétitif jusqu'au dernier élément du tableau, d'où utilisation de la boucle Pour... Faire. On doit initialiser la position du minimum à i puis faire le parcours du sous-tableau, dès qu'on

trouve un élément inférieur à ce minimum on change la position par l'indice de cet élément.

- Les paramètres formels de cette fonction sont **tf**, **nf** et **pd**

```
Fonction Entier premposmin (Entier nf, Entier tf :tableau [nf],
Entier pd)
Entier pm ← pd
Début
Pour j de pd+1 à nf Faire
   Si (tf[j] < tf [pm]) Alors
   pm ← j
   Finsi
Fin Pour
  Retourner (pm)
FinFct
```

5) **Spécification de la procédure permut :**

- **Résultat :** Permuter deux variables entiers **x** et **y**
- **Traitements :** pour permuter deux variables, on procède généralement à utiliser une variable intermédiaire et faire la permutation.
- Les paramètres formels de cette procédure sont **x** et **y**.

```
Procédure permut (E/S Entier x, y)
Entier int
Début
   int ← x
   x ← y
   y ← int
FinProc
```

7.1.4) Le tri à bulle (bubble sort)

L'appellation dite "tri à bulle" revient à la manière dont se déplacent les éléments dans le tableau, qui se ressemblent à la

manière des bulles qui remontent à la surface : plus elles sont grosses, plus elles remontent rapidement. Le principe de cette méthode consiste à comparer les éléments 2 à 2 : chaque élément est comparé avec son voisin d'indice immédiatement supérieur, et s'il y a lieu, ils sont permutés. Ceci permet de garantir le positionnement du plus grand élément en dernière position du tableau, même si d'autres permutations ont aussi été effectuées pour rapprocher le tableau de son état trié (d'autres bulles ont aussi monté vers la surface, sans toutefois y parvenir). Il suffit alors de recommencer sur le tableau dépourvu de son dernier élément. Si aucune permutation n'est effectuée, c'est que le tableau est déjà trié, on pourrait arrêter prématurément l'algorithme de tri.

Principe : Cette méthode de tri consiste à :

- **Comparer** les éléments du tableau **T** deux à deux,
- **Permuter** les contenus lorsque **l'ordre n'est pas respecté,**
- **Refaire** les actions **1** et **2** et ainsi de suite jusqu'à avoir finalement un tableau trié.

- **Exemple :**

Soit un tableau T contenant les dix éléments suivants :

T :	12	10	0	-5	8	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

Soit une variable booléenne **test** initialisée à vrai et qui devienne faux à chaque permutation.

➥ **Passage1 :**

T :	12	10	0	-5	8	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

Remarque : **T[1] > T[2] alors permutation**

T :	10	12	0	-5	8	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

Remarque : **T[2] > T[3] alors permutation**

T :	10	0	12	-5	8	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

Remarque : **T[3] > T[4] alors permutation**

T :	10	0	-5	12	8	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

Remarque : **T[4] > T[5] alors permutation**

T :	10	0	-5	8	12	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

Remarque : **T[5] = T[6] alors pas de permutation**

T :	10	0	-5	8	12	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

Remarque : **T[6] > T[7] alors permutation**

T :	10	0	-5	8	12	-2	12	2	40	-1
	1	2	3	4	5	6	7	8	9	10

Remarque : **T[7] > T[8] alors permutation**

T :	10	0	-5	8	12	-2	2	12	40	-1
	1	2	3	4	5	6	7	8	9	10

Remarque : **T[8] > T[9] alors pas de permutation**

T :	10	0	-5	8	12	-2	2	12	40	-1
	1	2	3	4	5	6	7	8	9	10

Remarque : **T[9] > T[10] alors permutation**

T :	10	0	-5	8	12	-2	2	12	-1	40
	1	2	3	4	5	6	7	8	9	10

Puisqu'on a atteint la fin du tableau et le contenu de la variable **test** est **Faux**, alors on doit recommencer un nouveau passage et ainsi de suite jusqu'à ce qu'on fasse un passage complet du tableau sans modifier le contenu de **test (Vrai)**.

1) **Spécification du programme principal du tri :**
- **Résultat :** Afficher le tableau T trié en utilisant une procédure **Affiche**
- **Traitements :** Il faut trier le tableau T, en utilisant une procédure **Tri**
- **Données :** Il faut remplir le tableau T et saisir la taille "n" en utilisant la procédure **Saisie**

```
Algorithme principal
  Début
    Saisie (n, T)
    Tri (n, T)
    Affiche (n, T)
Fin
```

2) **Spécification de la procédure Tri :**

- **Résultat :** Trier le tableau **Tf**
- **Traitements :** Il s'agit d'un traitement répétitif jusqu'à ce que la valeur de la variable test reste à vrai, d'où on utilise la structure Répéter ... Jusqu'à
- Le passage du tableau est un parcours du premier élément jusqu'au dernier élément, d'où l'utilisation de la boucle Pour ... Faire. Ensuite, il s'agit de comparer les paires des éléments consécutifs, s'ils ne sont pas dans le bon ordre via la permutation. Donc, on fera appel à une procédure intitulée "**permut**" permettant de permuter deux éléments. Les paramètres formels pour cette procédure sont le tableau "**tf**" et sa taille "**nf**".

```
Procédure Tri (Entier nf, E/S Entier tf :tableau[nf] )
Booléen test, i
Début
  Répéter
    test ← Vrai
      Pour i de 1 à tf-1 Faire
        Si (tf[i] > tf[i+1]) Alors
          Permut (tf[i], tf[i+1])
          Test←Faux
        Finsi
      FinPour
    Jusqu'à (test = Vrai)
FinProc
```

Note :

- Le problème de l'algorithme présenté plus haut est son coût : on recherche le plus petit élément parmi les n éléments, puis parmi les n-1 restants après le premier rangement, puis parmi n-2 etc. Au total il faut effectuer $n + (n-1) + ... + 1$ recherches, donc le nombre d'opérations peut être considéré comme proportionnel au carré du nombre de données à trier : $n + (n-1) + ... + 1 = n(n+1)/2$.
- Pour réduire ce coût, il s'agit de séparer le tableau en 2 sous-tableaux, ensuite d'effectuer un tri sur chacun de ces 2 sous-tableaux, enfin de les fusionner ensemble. Notons que pour effectuer le tri de chacun de ces 2 sous-tableaux, on applique récursivement cette même stratégie. Une autre stratégie basée sur le même principe consiste à faire un découpage en sous-tableaux de telle sorte que dans le premier tableau tous les éléments soient inférieurs à ceux du second tableau...

7.1.5) Le tri par fusion

L'idée de cette méthode est la suivante : Pour trier un tableau tab de n éléments, on le scinde en deux tableaux de même taille (à un élément prés). On les notes tab1 de taille n1 et tab2 de taille n-n1. Ces deux tableaux sont ensuite triés (appel récursif) et enfin fusionnés de manière à reformer le tableau t trié. La figure ci-dessous reprend l'exemple du tri par sélection et montre comment le tri par fusion fonctionne à travers les étapes numérotées de 1 à 21.

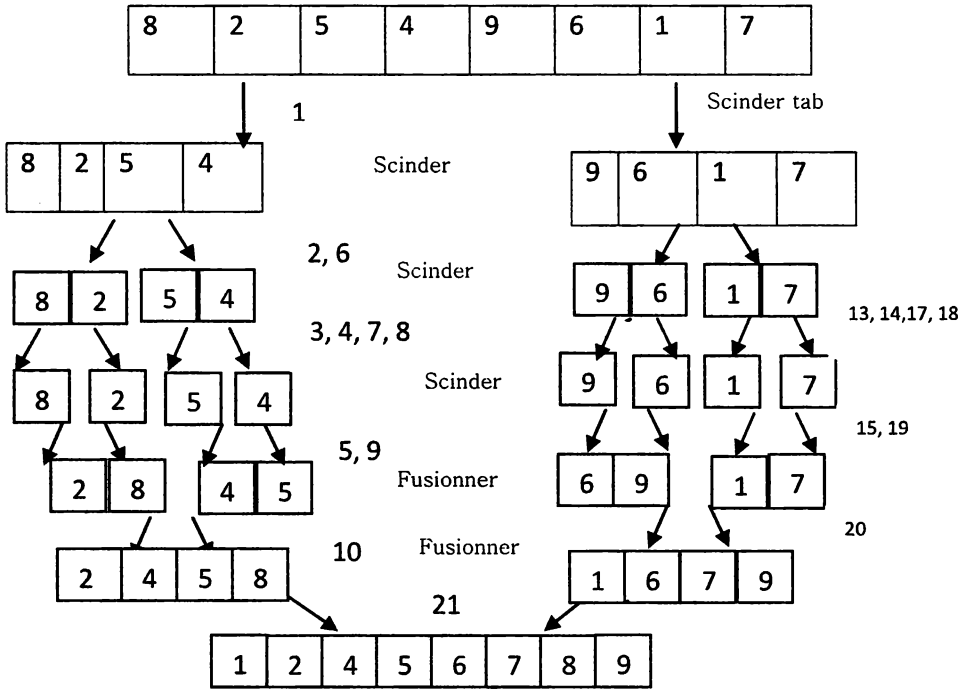


Figure 11. Les étapes d'un exemple de tri par fusion

Pour réaliser le tri par fusion on a besoin des fonctions suivantes à savoir :

- La procédure "scinder (**Entier** tab, **Entier** n, **Entier** tab1, **Entier** n1, **Entier** tab2)" : procédure qui permet de copier les n1 premiers éléments du tableau tab dans un tableau tab1 et le reste dans un tableau tab2.
- La fonction "Concatener (Entier tab1, Entier n1, Entier tab2, Entier n2, Entier i2)" : Copie le tableau "tab2" de taille "n2" à la fin du tableau "tab1" de taille initiale "n1". La copie débute à l'indice "i2" dans "tab2". Après la copie, la nouvelle taille de "tab1" est retournée par la fonction.

- La procédure "fusionner (**Entier** tab, **Entier** tab1, **Entier** n1, **Entier** tab2, **Entier** n2)" : Recopie les éléments des tableaux tab1 et tab2 dans le tableau t de façon à ce qu'ils soient triés. Les éléments de tab1 et de tab2 sont supposés triés.
- La procédure "triefusion (**Entier** tab, **Entier** n)" : Trie les n éléments du tableau tab par la méthode de tri par fusion.

➡ Procédure Scinder

```
Procedure scinder (Entier tab : tableau[], Entier n, S Entier
tab1:tableau[], Entier n1, S Entier tab2 :tableau[])
Déclarations locales
 Entier i, j
 Debut
 Pour i de 1 à n1
  Faire
  tab1[i]←tab[i]
  Finpour
  Pour j de i à n2
   Faire
    tab2[j]←tab[i]
     i←i+1
  Finpour
Fin
```

➡ Fonction concatener

```
Fonction Entier Concatener (Entier tab1 : tableau[], Entier n1,
Entier tab2:tableau[], Entier n2, Entier i2)
Entier i
Début
Pour i de 1 à n2
 Faire
  tab1[n1]←tab2[i2+1]
  n1←n1+1
 Finpour
```

Retourner (n1)
Fin

➥ **Procédure Fusionner**

Procédure Fusionner (**S** Entier tab : tableau[], Entier tab1 tableau[], Entier n1, Entier tab2 tableau[], Entier n2)
Entier i1, i2, i
Début
i←1
Pour i1←1 ET i2←1 à i1=n1 ET i2=n2
Faire
Si (tab1[i1]<tab2[i2])
Alors
tab[i] ←tab1[i1]
Sinon
tab[i] ←tab2[i2]
Finsi
i←i+1
Finpour
i←concatener(tab, i, tab1, n1-i1, i1);
concatener(tab, i, tab2, n2-i2, i2) ;
Fin

4) Procédure triefusion : On suppose que la procédure "vider" est une procédure déjà prédéfinie et qui permet de vider le contenu d'un tableau et que la fonction "estVide" désigne une fonction booléenne qui retourne si un tableau est vide ou non.

Procedure trierfusion (**Entier** n, **E/S Entier** tab :**tableau** [])
Entier n1, n2, tab1 : **tableau** [n1], tab2 : **tableau** [n2]
Début
Si (n>1) **Alors**
n1←n/2
n2←n-n1
Si (estVide(tab1)=Faux **ET** estVide(tab2)=Faux) **Alors**
Débutalors
scinder(tab, n, tab1, n1, tab2) ;

```
    trierfusion(tab1, n1) ;
    trierfusion(tab2, n-n1) ;
    fusionner(tab, tab1, n1, tab2, n-n1) ;
    vider(tab1) ;
    vider(tab2) ;
   Finalors
   Sinon // Erreur pas d'espace mémoire
   Si (estVide(tab1)=Faux) Alors
   vider(tab1)
   Finsi
    Si (estVide (tab2)=Faux)) Alors vider (tab2)
    Finsi
  Finsi
Fin
```

Remarque : Le principe de la méthode précédente est très intéressant, puisqu'il permet de réduire de manière significative le nombre d'opérations à effectuer. Néanmoins, le coût des fonctions de séparation et de fusion est encore trop important. En particulier, la fusion exige la comparaison des premiers éléments de chaque sous-tableau trié.

7.2) Les méthodes de recherches dans un tableau

La recherche d'un élément dans un tableau ou dans une liste de valeur est un traitement très utile en informatique. Parmi les méthodes de recherches, on cite :

- La recherche séquentielle,
- La recherche dichotomique.

7.2.1) Recherche séquentielle

Principe : Cette méthode de recherche consiste à parcourir les éléments du tableau un par un jusqu'à trouver la valeur cherchée ou atteindre la fin du tableau.

Exemple : Soit un tableau T contenant les dix éléments suivants :

T :	12	10	0	-5	8	12	-2	2	40	-1
	1	2	3	4	5	6	7	8	9	10

- Pour X = -2 le programme affichera "-2 existe dans le tableau" ;
- Pour X = 5 le programme affichera "5 n'existe pas dans le tableau".

1) **Spécification du programme principale :**
- **Résultat :** Afficher le résultat de l'existence de X dans le tableau T en utilisant une procédure **Affiche**
- **Traitements :** Il faut vérifier si X existe dans le tableau T ou non, en utilisant une procédure **Recherche**
- **Données :** Il faut remplir le tableau T, saisir la taille N et la valeur X à chercher, en utilisant la procédure "**Saisie**".

Algorithme principal :
Début
  Saisie (N, T, X)
  verif ← **Recherche** (N, T, X)
  Affiche (verif)
Fin

2) **Spécification de la procédure Saisie :**

- **Résultat :** Saisir **nf, xf** et remplir **tf**
- **Traitements :** Le remplissage d'un tableau est une action répétitive, ou on connaît le nombre de répétitions qui est égale à "**nf**", d'où utilisation de la boucle "pour". La saisie de l'entier "**nf**" doit être contrôlée pour ne pas saisir un entier négatif ou supérieur à 100. Cette procédure admet trois paramètres formels qui sont "**nf**", "**xf**" et "**tf**".

```
Procédure Saisie (S Entier nf, E/S Entier tf : tableau[nd] , S Entier xf)
Début
 Répéter
  Ecrire ("Donner la taille du tableau : ")
  Lire (nf)
  Jusqu'à (nf>1 ET nf<100)
  Pour i de 1 à nf Faire
   Ecrire ("Donner l'élément N° Entier :", i)
  Lire (tf[i])
Fin Pour
Ecrire ("Donner la valeur à chercher : ")
Lire (xf)
FinProc
```

3) **Spécification de la fonction Recherche :**

- **résultat :** Retourner un résultat booléen déterminant l'existence.
- **Traitements :** Il s'agit de comparer "**xf**" avec chaque élément du tableau "**tf**" jusqu'à trouver la valeur ou atteindre la fin du tableau, donc utilisation d'une structure itérative à condition d'arrêt et au minimum faire une comparaison si on trouve la valeur à la première case du tableau, d'où utilisation de la boucle Répéter ... Jusqu'à.

- Les paramètres formels de cette fonction sont "**xf**", "**nf**" et "**tf**".

Fonction Booléen recherche (**S Entier** nf, **E/S Entier** tf : **tableau**[nd] , **S Entier** xf)
Booléen trouve ← Faux, i ←1
Répéter
Si tf[i] = xf **Alors**
trouve ←Vrai
Sinon
i ←i+1
Finsi
Jusqu'à (trouve = Vrai) **OU** (i > nf)
Retourner (trouve)
FinFct

4) **Spécification de la procédure affiche :**
- **Résultat :** Afficher un commentaire pour dire si la valeur existe dans le tableau ou non.
- **Traitements :** Il s'agit d'une structure conditionnelle suivant la valeur de "**veriff**" qui affiche un commentaire.
- Les paramètres formels de cette procédure sont "**veriff**"

Procédure affiche (Boolen veriff)
Début
Si (Veriff = Vrai) **Alors**
Ecrire ("L'entier existe dans le tableau")
Sinon
Ecrire ("L'entier n'existe pas dans le tableau")
Finsi
FinProc

7.2.2) Recherche dichotomique

L'approche dichotomique de recherche de l'information dans un tableau procède avec le principe suivant :
Principe : on compare l'élément recherché à celui du milieu du tableau. S'il y a égalité, on a terminé, sinon on peut déterminer dans quel sous-tableau il est utile de chercher, et on n'a plus que la moitié des données à parcourir. On itère ce processus jusqu'à ce qu'on trouve la donnée ou jusqu'à ce qu'on échoue sur un tableau de 1 seul élément :
Fonctionnement : exemple de recherche d'un entier parmi 40000.

- Au départ, on cherche un entier parmi 40 000.
- Après le test n°1, on ne le cherche plus que parmi 20 000.
- Après le test n°2, on ne le cherche plus que parmi 10 000.
- Après le test n°3, on ne le cherche plus que parmi 5 000.
- etc.
- Après le test n°15, on ne le cherche plus que parmi 2.
- Après le test n°16, on ne le cherche plus que parmi 1.

```
Fonction Entier rechercheDicho (Entier tab:tableau[], Entier
n, Entier element)
Déclarations locales
Entier debut←1, fin←n, pos←-1, milieu←n/2
Début
 Tantque ((debut≤fin) et (tab[milieu]<>element))
 Faire
   Si (tab[milieu]=element) Alors
    pos ← milieu ;
    Sinon
     Debutsinon
     Si (tab[milieu]>element) Alors
```

```
        fin ←milieu-1
        Sinon
        debut ← milieu+1;
       Finsi
       milieu ← (debut+fin)/2 ;
     Finsinon
    Finsi
 Fintantque
 Retourner (pos);
FinFct
```

- **Application 1 :** *donner le pseudo-code* ***des algorithmes de tri suivant :***
  1) *Le tri par permutation*
  2) *Le tri par comptage*
  3) *Le tri alphabétique*

- **Solution :**

  1) *Tri par permutation*

```
Pour i de de 1 à 9
Faire
Si (tab [i+1] < tab [i]) Alors
Alors
  Abouger←tab [i+1]
  j←1
 Tanque ((j < i) et (tab [j] < tab [i+1]))
  Faire
  j ←j+1
 FinTantque
Pour k de de i+1 a j+1 par pas -1
 Faire
 Tab [k]← tab [k-1]
 Finpour
  Tab [j]← abouger
 Finsi
```

Fin pour

2) Tri par comptage

```
Pour i de 1 à 10
  Faire
   Res [i] ←0
   Nb [i] ← 0
  Pour j de 1 à 10
    Faire
    Si (Tab [J] < Tab [I]) Alors
     Nb [i] ←Nb [i] + 1
   Finsi
   Finpour
  Finpour
  Pour i de 1 à 10
     Faire
       j ← Nb [i]
      Tantque Res [j] <> 0
        Faire
       j ←j+1
      Fintq
        Res [j] ←Tab [i]
  Finpour
```

3) Le tri alphabétique

```
Pour Nbmots de 1 à 10
 Faire
 Ecrire ("Entrer Le Mot Suivant")
  Lire (Mot)
 Pluspetit← Vrai
 J ←1
 Tantque (Pluspetit Et (J < Nbmots))
  Faire
```

```
 K ←1
 Tantque ((Mots [J, K] = Mot [K]) Et K <= 20)
  Faire
   K ←K+1
  FinTantque
 Si (Mots [J, K] < Mot [K])
  Alors
  J ←J+1
  Sinon
  Pluspetit ←Faux
 Finsi
FinTantque
Si (J < Nbmots)
Alors
Pour I de Nbmots à J+1 par pas de -1
Faire
Pour K de 1 à 20
Faire
Mots [I, K] ←Mots [I-1, K]
Finpour
Finpour
Finsi
Pour K de 1 à 20
Faire
Mots [J, K] ←Mot [K]
Finpour
Finpour
Pour I de 1 à Nbmots
 Faire
 Ecrire (Mots [I])
Finpour
```

Examens corrigés

Examen 1

Exercice 1 (1,5 pt + 2,5 pts + 7 pts)

On se propose d'écrire un algorithme permettant, à partir de deux tableaux d'entiers T1 et T2 contenant chacun N entiers, de modifier les tableaux de telle sorte que T1 contiendra tous les entiers impairs contenus à l'origine dans T1 et T2. On suppose que si T1 et T2 sont réunis ensemble, ils contiennent N entiers pairs et N entiers impairs.

Par exemple, si au départ T1=[0,7,5,3] et T2=[4,2,9,6], alors à la fin de l'algorithme, T1=[0,4,2,6] et T2=[7,5,9,3].

Sachant que le principe du traitement se base essentiellement sur la permutation d'éléments entre T1 et T2, élaborer les trois éléments de réponse suivants :

a) Description du rôle de chacune des données de l'environnement (Les différentes données utilisées) de votre algorithme.
b) Description sous forme de tirets de la logique de traitement à laquelle tu opteras.
c) L'algorithme complet qui assure le traitement décrit ci-dessus.

NB : On suppose que les deux tableaux T1 et T2 sont disponibles et ne nécessitent pas la saisie.

Exercice 2 (2 pts + 5 pts + 2 pts)

On se propose d'écrire un algorithme qui permet de convertir un nombre de la base binaire (2) en son équivalent valeur dans la base 10.

Exemple : rappelant la conversion de la base 2 vers la base 10 :

$(100110)_2 = 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 38$

On suppose que le nombre qui est dans la base deux, est formé de 64 bits et qu'il est représenté par un tableau de caractères où chaque case de ce dernier, contient obligatoirement le caractère '1' ou '0'.

a) Décrire l'environnement (Les différentes données utilisées) de votre algorithme en précisant le nom, le type et le rôle de chacune des données et en supposant que pour le type Entier on réserve 04 Octets et pour le type Réel on réserve 08 Octets.

b) Rédiger votre solution algorithmique qui assure la conversion demandée.

c) Et enfin, élaborer la trace d'exécution pour l'exemple cité ci-dessus.

Corrigé de l'examen n°1

Exercice 1

a)

L'environnement de l'algorithme est constitué de :

D.E/S : Entier T1, T2 : tableau [N]

T1 et T2 sont les deux tableaux d'entiers qui initialement possèdent N entiers et à la fin de l'algorithme ils contiendront respectivement les N entiers impaire et les N entiers impaires.

D.I: Entier ind_p, ind_i, Temp, i, j, k

ind_p et ind_i sont deux données internes qui indiquent les indices des cases à permuter entre T1 et T2.

Temp est une donnée temporaire assure la permutation entre deux cases de T1 et T2.

i, j et k est un indice qui permet le parcours des deux tableaux en même temps.

b)

- Prévoir une boucle qui itère tant que le nombre des permutations dans l'un des sens n'a pas atteint N.
- A l'intérieur de la première boucle, on prévoit une boucle qui parcourt T1 à la recherche des éléments impairs, à la rencontre de chacun on marque son indice dans ind_i.

- A l'intérieur de la première boucle, on prévoit une boucle qui parcourt T2 à la recherche des éléments pairs, à la rencontre de chacun on marque son indice dans ind_p.
- Une fois un élément de T1 et T2 ont été marqués, on réalise la permutation et on réitère via la première boucle.

b) L'algorithme complet qui assure le traitement décrit ci-dessus.

```
D.E/S : Entier T1, T2 : tableau [N]
D.I: Entier ind_P, ind_I, temp, i, j, k
     Booléen trouve1, trouve2
Début
 i←0
 j←0
  trouve1←Faux
  trouve2←Faux
   TantQue( i<N ET (j<N)) Faire
    Si(T1[i]%2<>0) Alors
     trouve1←Vrai
      ind_I←i
       Sinon
       i←i+1
      Finsi
```

```
        Si (T2[j]%2<>0) Alors
        Débutalors
        trouve2←Vrai
        ind_P←i
         Finalors
         Sinon
         j←j+1
        Finsi
        Si (trouve1 ET trouv2) Alors
          temp←T1[ind_I]
         T1[ind_I]←T2[ind_P]
         T2[ind_P]←temp
         trouve1←Faux
          trouve2←Faux
          i←i+1
           j←j+1
        Finsi
FinTantque
Fin
```

Exercice 2

a)

D.E :

- **Caractère** nb_2 : tableau [64] : Tableau de caractères de taille 64 dans lequel on trouve la représentation du nombre en binaire.

D.S :

- **Réel** nb : est un réel qui contient le résultat. Il est sur 8 octets car le nombre binaire est sur 64 bits

D.I :

- **Entier** i : est l'indice qui parcourt le tableau de 64 cases
- **Réel** deux_puis_i : contient à chaque itération la valeur calculée de 2 à la puissance i. elle est choisie réelle car on a la possibilité d'atteindre 2^{63}

b)

D.E : **Caractère** *nb_2 : tableau [64]*

D.S : **Réel** *nb*

D.I : **Entier** *i*

Réel *deux_puis_i*

Début

deux_puis_i← 1

nb ← 0

Pour i de 63 à 0 par pas de -1

Faire

Si (nb_2[i] = '1') **Alors**

nb ←nb + deux_puis_i

Finsi

deux_puis_i ← deux_puis_i * 2

FinPour

Fin

c) Et enfin, élaborer la trace d'exécution pour l'exemple cité ci-dessus.

Une trace correcte mérite 2 pts

Moyenne : 1 pt

Et Fausse Zéro pt

Examen 2

Exercice 1 (5.5 points)

Q1) Cocher la réponse la plus significative : **(2 points)**

Une variable est une ☐ donnée qui change ☐ donnée qui ne change pas ☐ zone mémoire dont la valeur varie	**Une valeur numérique peut être** ☐ de type entier ☐ de type réel ☐ de type chaîne de caractères
L'instruction qui permet d'attribuer une valeur à une variable est: ☐ l'affectation, ☐ la lecture ☐ l'écriture	**Soit S une variable de type chaîne de caractères** ☐ S← 0,2561 ☐ S← "Tunisie" ☐ S← "0,2561"

Note : Les réponses doivent figurer sur la copie d'examen rendue.

Q2) Compléter l'algorithme suivant pour calculer la surface et le diamètre d'un disque **(1.5 points)**

Algorithme Surface_Disque_diamètre

Constante

.....

Variables

.....

.....

Début

.....

.....

.....

Surface⟵ PI*Rayon*Rayon ;

Ecrire ('la surface du disque est :.....', Surface) ;

Ecrire ('le diamètre du disque est :.....', diamètre) ;

Fin

Q3) Un magasin de reprographie facture 0.05 dinar les dix premières photocopies, 0.04 dinar les vingt suivantes et 0.03 au-delà de 30. Écrire un algorithme qui demande à l'utilisateur le nombre de photocopies effectués et qui affiche la facture correspondante. **(2 points)**

Exercice 2 (8 points)

On considère la suite d'instructions suivantes :

```
x ←0
y←x+2
z←1
TantQue (z<5) faire
        z←z+1
        x←z+3
        y←y*2
FinTantQue
```

Q1) Dresser un tableau pour préciser la trace d'exécution pour chaque itération exécutée par l'algorithme plus haut (identifiez les valeurs de x, y et z).

Soit les variables entières A, B, C et D, on considère la suite d'instructions suivantes :

```
Si (A>5 OU B>C) ET (D>A OU D<6) Alors
    Débutalors
     A← A-B
     D ← B+C
    Finalors
    Sinon
    Débutsinon
     C← A-B
     Si (C>3) Alors
          D← -D
          B←0
     Fin Si
     Finsinon
Fin Si
```

Q2) Donnez les valeurs finales de A, B, C et D après exécution des cette suite d'instructions en considérant les valeurs initiales suivantes :

	A	B	C	D
a)	5	3	4	6
b)	8	1	0	2

Exercice 3 (7.5 points)

On veut écrire les algorithmes permettant de manipuler un ensemble d'opérations sur un/des entier(s).

A) Ecrire un algorithme qui compte le nombre de chiffres 0 contenu dans la valeur d'un entier X. L'affichage de votre algorithme ressemblera à l'exemple suivant :

Le nombre des zero dans l'écriture de X=1054008 est 3.

B) L'objectif est de calculer l'entier X obtenu à partir d'un entier Y en supprimant le chiffre 0 de l'écriture de X.

Exemples :

- Si X=1054008, alors Y=1548.
- Si X=227, alors Y=227.

On utilise la démarche suivante :

- On calcule l'entier obtenu en inversant l'ordre des chiffres de X et en supprimant les 0.
- On inverse à nouveau les chiffres du nombre obtenu.

Exemple :

Si X=1054008 :

- On calcule d'abord l'entier 8451.
- Puis on inverse ses chiffres pour obtenir l'entier Y=1548.

Question : Ecrire le pseudocode de cet algorithme.

Note : N'utiliser pas les tableaux ni les chaînes de caractères

Corrigé de l'examen n°2

Exercice 1 (5 points)

Q1) Cocher la réponse la plus significative : (1.5points)

Une variable est une [x] donnée qui change [ ] donnée qui ne change [x] zone mémoire dont la valeur varie	**Une valeur numérique peut être** [x] de type entier [x] de type réel [ ] de type chaîne de caractères
L'instruction qui permet d'attribuer une valeur à une variable est: [x] l'affectation, [x] la lecture [ ] l'écriture	**Soit S une variable de type chaîne de caractères** [ ] S← 0,2561 [x] S← "Tunisia" [x] S← "0,2561"

Q2)

Algorithme Surface_Disque_diamètre ;
Constante
Const Réel PI=3.14..... (0.25)
Variables
D.E : Réel Rayon ;.....(0.25)
D.S : Réel Surface, diamètre ;...(0.25)
.....
Début
Ecrire ('Donner le rayon :') ;.....(0.25)

Lire(Rayon) ;..... (0.25)
diamètre ←2**Rayon ;..... (0.25)*
Surface ←PI*Rayon*Rayon
Ecrire ('la surface du disque est =**Réel** :', Surface)
Ecrire ('le diamètre du disque est =**Réel**:', diamètre)
Fin

Q3)

```
Algorithme reprographie
  D.E : Entier nbphoto
  D.S : Entier facture
  Début
   Ecrire ('Donner le nombre de photocopies') ;
   Lire (nbphoto) ;
   Si(nbphoto<=10) Alors
     facture←0.05*nbphoto ;
     sinon
     Si (nbphoto<=30) Alors
      facture←0.04*(nbphoto-10)+10*0.05 ;
      Sinon
     facture←0.04*(nbphoto-30)+10*0.05+20*0.04 ;
    Finsi
   Finsi
   Ecrire('La facture correspondante est définie par',facture) ;
  Fin
```

Exercice 2 (8 points)

On considère la suite d'instructions suivantes :

x ←0
y←x+2

```
z←1
TantQue (z<5) Faire
        z←z+1
        x←z+3
        y←y*2
Fin TantQue
```

Q1)

	x	y	z
Valeurs initiales	0	2	1
Itération 1	5	4	2
Itération 2	6	8	3
Itération 3	7	16	4
Itération 4	8	32	5

Q2)

	A	**B**	**C**	**D**
a) val initiale	5	3	4	6
Val finale	inchangée	inchangée	2	inchangée
b) val initiale	8	1	0	2
Val finale	7	inchangée	inchangée	1

Exercice 3 (7 points)

A)

Algorithme **comptchiff**
D.E :Entlong X, X_depart;
D.S :Entier nb_zero←0
Début
Ecrire('Donner X>0')
Lire(X)
X_depart←X
Tantque(X <> 0)
Faire
Si(X%10=0) **Alors** //le dernier chiffre de X est 0 ; on incrémente nb_zero
nb_zero←nb_zero+1
Finsi
X=X/10;// on tronque le dernier chiffre de l'écriture de X
FinTantQue
Ecrire ("Le nombre de zero dans l'écriture de **Entier** est **Entier**",X_depart,nb_zero)
Fin

B)

Algorithme Supprimzero
D.E : Entlong X, X_depart;
D.S : **Entlong** Y←0;
Début
Ecrire('Donner X>0') ;
Lire(X) ;
X_depart ←X ;
Tantque(X <>0)
Début
Si(X%10<>0) **Alors**

```
  Y←10*Y +(X%10);
  X←X/10;
Finsi
Fintantque
Ecrire ("Etape1 achevé : Entier", Y); // réinitialisation de Y et X
X←Y; Y←0;
TantQue(X <> 0)
Faire
 Si(X%10<>0) Alors
  Y←10*Y +(X%10);
Finsi
X=X/10;
Fintantque
Ecrire("Etape2 achevé : X= Entier et Y= Entier",X_depart,Y);
Fin
```

Examen 3

Exercice 1 (8 points)

Le but de cet exercice est de calculer la moyenne d'un étudiant dont l'identifiant est saisi au clavier.

- Soit une classe de première année TI incluant 25 étudiants et dont chaque étudiant est identifié par un numéro allant de 1 à 25. Chaque étudiant dispose de 10 notes.

Nous proposons de représenter ces informations sous forme d'un tableau T à 2 dimensions (matrice). L'accès à la case Notes[i][j] de la matrice Notes permet d'obtenir la note j de l'étudiant i. On vous demande de répondre aux questions suivantes :

Q1) Faire les déclarations des variables et des structures nécessaires pour la résolution de ce problème.

Q2) Supposons que la matrice est déjà remplie, rédiger un algorithme permettant de calculer la moyenne d'un étudiant donné (dont le numéro est précisé par l'utilisateur au clavier).

Q3) Étendre (compléter) l'algorithme de la question Q2 de façon à ce que l'utilisateur puisse, après avoir obtenu la moyenne d'un étudiant, demander la moyenne d'un nouvel étudiant. L'algorithme termine l'exécution lorsque l'utilisateur entre comme réponse la valeur n ou N (non).

Exemple d'exécution :

Donner le numéro de l'étudiant dont vous voulez calculer la moyenne : 1

La moyenne de l'étudiant n°1 est=10.8

Voulez vous calculer la moyenne d'un autre étudiant Oo | Nn?
O

Donner le numéro de l'étudiant dont vous voulez calculer la moyenne : 6

La moyenne de l'étudiant n°6 est=12

Q4) Écrire un algorithme permettant de calculer les moyenne te tous les étudiants afin de les enregistrer dans un tableau "**Moyennes**" de taille 25. Puis, choisir un algorithme de tri de votre choix pour trier le tableau "**Moyennes**" dans l'ordre croissant. Enfin, afficher l'identifiant (indice) de l'étudiant ayant une moyenne maximale.

Exercice 2 (6 points)

On rappelle l'algorithme de la décomposition binaire pour un entier n≥0, par exemple pour n=18 :

18=9*2+0

9=4*2+1

4=2*2+0

2=1*2+0

1=0*2+1

On a alors : $n=18=(10010)_2$

On vous demande de faire les algorithmes suivants :

Q1) Écrire un algorithme calcule_long_decomp_bin qui permet de saisir un entier positif de taille n et qui retourne la longueur de sa décomposition binaire. S'assurer que l'algorithme redemande à l'utilisateur d'entrer un entier au clavier en cas de la

saisie d'un entier négatif. Par exemple, si n=18, l'algorithme affichera 5.

Q2) On veut enregistrer la décomposition binaire d'un entier dans un tableau d'entier nommé decomp. On supposera le tableau de taille L+1 où L est la longueur de décomposition binaire de n. De même, on placera l'entier -1 à la fin de la décomposition binaire de n.

Par exemple : Si n=18 et L= 5 :

decomp :

0	1	0	0	1	-1
1	2	3	4	5	6

Réécrire et compléter l'algorithme la question Q1 afin de calculer la décomposition binaire d'un entier, et de placer les bits de cette décomposition dans le tableau decomp, ensuite de placer -1 à la fin de cette décomposition.

Q3) Réécrire et compléter l'algorithme de la question précédente afin d'afficher la décomposition binaire enregistré dans le tableau decomp. *Par exemple, si* le tableau decomp contient la décomposition binaire de n=18, l'affichage obtenu sera : $13=(10010)_2$

Q4) Soit un entier n dont la décomposition binaire de longueur l s'écrit sous la forme :

$$n=\sum_{i=0}^{l-1} b_i * 2^i$$

Soit x un réel, alors x^n peut s'écrire sous la forme :

$$X^n=\prod_{i=1}^{l} X^{2^{(i-1)}}$$

A partir des valeurs contenues dans le tableau comp (supposé rempli) et un réel saisie au clavier, déduisez un algorithme qui calcule x^n à l'aide de la décomposition binaire de n. Par exemple, pour n=18 et un réel *x*, supposons que la décomposition de n est déjà stockée dans le tableau suivant et les bits 1 sont présents dans les positions 2 et 5.

0	1	0	0	1	-1

$$X^{18}=X^{(2^1)} * X^{(2^4)}$$

Exercice 3 (6 points)

Écrire un algorithme qui permet de saisir et de valider 4 codes de choix de filière d'orientation pour les étudiants de deuxième année TI. On suppose que l'on dispose de 10 codes différents qui sont stockés dans un tableau trié. La validation des codes de choix est répétée tant que le choix des 4 codes valides n'a pas encore achevé. A chaque fois qu'un code est validé, doit être enregistré dans un tableau **TabChoix**. La validation des codes saisis s'effectue en les comparants des codes déjà stockés dans un tableau d'orientation **TabOrien.**

Exemple :

TabOrien

000 1	0011	1201	1202	1100	1240	1300	1210	0012	1502

TabChoix

1201	0001	1210	1100

Corrigé de l'examen n°3

Exercice 1 (8,5 points)

Q1) 1 point

Déclaration des variables **(4*0,25=1 point)**

D.E : Réel Notes:tableau[25][10] ;

D.S : Réél MoyE, SomN, numE

Q2) (2.5 points)

```
Algorithme Moyenne_Etudiant
D.E : Réel Notes :tableau[25][10]
D.S : Réel MoyE, SomN, numE
D.I : Entier i
Début
  Ecrire (''donner le numéro de l'étudiant dont vous voulez
calculer la moyenne'')
  Lire (numE)
  SomN←0
  Pour i de 1 à 10 faire
    SomN←SomN+Notes[numE][i];
  Finpour
  MoyE←SomN div 10
    Ecrire ("La moyenne de l'étudiant numE est=MoyE",
numE, MoyE)
Fin
```

Q3) (2 points)

Algorithme Moyenne_Sollicitee
D.E : Réel Notes : **tableau** [25][10] ; **Entier** numE
D.S : Réel MoyE, SomN
D.I : Entier i ;
Booléen Réponse
Début
Réponse←oui
Tant que (Réponse="oui")
Faire
Ecrire ("donner le numéro de l'étudiant dont vous voulez calculer la moyenne")
Lire (numE)
SomN←0
Pour i de 1 à 10 **faire**
SomN←SomN+Notes[numE][i]
Finpour
MoyE←SomN div 10
Ecrire("La moyenne de l'étudiant entier est un réel", numE, MoyE)
Ecrire (« Voulez vous calculer la moyenne d'un autre étudiant oO|nN ? »)
Lire (Réponse)
FinTantque
Fin

Q4) (3 points).

Algorithme MoyenneVect
D.E : Réel Notes:**tableau**[25][10]; **Entier** numE
D.S : Réel SomN, Moyennes:**tableau**[25][10]

```
D.I : Entier i ; Réel vaux, MoyMax
Début
   Pour i de 1 à 25   Faire
    SomN←0
    Pour j de 1 à 10 faire
      SomN←SomN+Notes[i][j]
    Finpour
    Moyennes[i]←SomN div 10
  Finpour
 //Tri de séléction
 Pour i de 1 à 24 Faire
  Pour j de i+1 à 25 Faire
    Si (T[i]>T[j]) Alors
      Vaux←T[i]
      T[i]←T[j]
      T[j]←vaux
    Finsi
   Finpour
 Finpour
MoyMax←T[25]
Pour i de 1 à 25 faire
  Si(Notes[i]=MoyMax) alors
   Ecrire("L'étudiant d'indice  Entier possède la moyenne
maximale Réel ", i, MoyMax)
   Finsi
Finpour
Fin
```

Exercice 2 (7 points)

Q1)

```
Algorithme calcule_long_decomp_bin
D.E : Entier n
D.S : Entier L
D.I : Entier c
Début
 Répéter
  Ecrire (''Donner n>0'')
  Lire (n)
 Jusqu'à (n>0)
 L←0 ;
 Répéter
 n←n/2
 L←L+1
 Jusqu'à (n=0)
Ecrire (''L nombre n admet L décompositions binaires'', n, L)
Fin
```

Q2) (1.5 points)

```
Algorithme calcule_long_decomp_bin
D.E : Entier n
D.S : Entier L, Entier decomp:Tableau[L]
D.I : Entier c
Début
 Répéter
  Ecrire (''Donner n>0'')
  Lire (n)
 Jusqu'à (n>0)
 L←0 ;
 Répéter
```

```
 n←n/2
 L←L+1
 decomp[L]←n%2
Jusqu'à (n=0)
L←L+1 ;
decomp[L]←-1
Fin
```

Q3)

```
Algorithme calcule_long_decomp_bin
D.E : Entier n
D.S : Entier decomp:Tableau[L]
D.I : Entier c
Début
 Répéter
  Ecrire ("Donner n>0")
  Lire (n)
 Jusqu'à (n>0)
 L←0
 Répéter
 n←n/2
 L←L+1
 decomp[L]←n%2
Jusqu'à (n=0)
L←L+1
decomp[L]←-1
Ecrire ("La décomposition binaire du nombre n est :")
Pour i de L-1 à 1 Faire par pas de -1
 Ecrire (decomp[i]) ;
Finpour
```

Fin
Fin

Q4)

Algorithme calcule_Puiss_bin
D.E :Entier n ; **Réel** X; **Réel** decomp:**Tableau**[L]
D.S : Réel PuissR
D.I : Entier c
Début
Ecrire (''Donner un Réel X')
Lire (X)
Ecrire (''Donner La longueur de la décomposition binaire'')
Lire (L)
Ecrire (''Donner n'')
Lire (n)
PuissR←1
Pour i de 1 à L **Faire**
Si(decomp[i]=1) **Alors**
PuissR←PuissR*X^(2^(i-1))
Finsi
Finpour
Ecrire (''X puissance n='', PuissR)
Fin

Exercice 3 (4.5 points)

Algorithme ValidationCode
DE : Entier TabOrient[10], **Entier** code
D.S : Entier TabChoix[4]

```
D.I : Entier i,j ; Booléen trouve←faux
Début
j←1
Répéter
   Ecrire ("Saisir votre code ?")
   Lire (code)
     i←1
   TantQue(trouve=faux et i<10)
   Faire
   Si (TabOrient [i]=code) Alors
    trouve←vrai
   Finsi
   i←i+1
   Fintantque
  Si (trouve=vrai) Alors
   TabChoix [j]← code
   j←j+1
  Sinon
 Ecrire("Code saisi erroné")
 Finsi
Jusqu'à (j>4)
Fin
```

Examen 4

Exercice 1 (6 points)

I) Partie1 (Questions de cours)

a) Quel est l'intérêt d'écrire un algorithme avant de commencer à programmer ?
b) **b)** Pourquoi choisir l'instruction **Selon** plutôt que **Si** ?
c) Quand utiliser **Répéter..Jusqu'à** plutôt que **Tantque** ?
d) Quelle est la différence entre variable locale et variable globale ?
e) Quand utiliser l'instruction **Pour ?**

II) Partie 2

Écrire un algorithme itératif permettant de calculer U, la racine cubique de V, en utilisant la méthode de Newton.
Principe : En partant de l'expression $U_0=V+1/3(1/V-V)$ on calculera par récurrence : $U_{n+1}=U_n+1/3(V/U_n^2-U_n)$. Les itérations s'arrêtent lorsque l'erreur relative $e= (U_{n+1}-U_n)/U_n$ devient plus petite que la précision Pr=0.00001. On prend, alors, comme valeur approchée de U la valeur de U_{n+1}.
Exemple : Pour V=27 le programme fournit comme résultat U=3.
Remarque : Ne pas utiliser les fonctions **Exp** et **Log pour** calculer la racine cubique.

Exercice 2 (7 points)

a) Soit l'énoncé suivant:

- Un nombre n (strictement positif) est ***parfait*** s'il est égal à la somme de ses diviseurs (non compris le nombre n lui-même). Exemple: 6 est parfait car 6=1+2+3.
- Un nombre n est ***déficient*** s'il est strictement plus grand que la somme de ses diviseurs.
Exemple : 8 est déficient car 8>1+2+4.
- Un nombre est ***abondant*** s'il est strictement plus petit que la somme de ses diviseurs. Exemple : 12 est abondant car 12<1+2+3+4+6.

On vous demande d'écrire l'algorithme **testernombre** qui teste et affiche si le nombre reçu est déficient, parfait ou abondant.

b) Écrire l'algorithme qui détermine le chiffre minimum d'un entier positif saisi au clavier. On suppose que le nombre des chiffres est non connu à l'avance par l'utilisateur et par conséquent, il saisit autant de chiffre qu'il veut.

Exemple : Chiffre minimum du nombre **2458169** est **1.**

Exercice 3 (7 points)

1) Indiquer, dans les cases correspondantes de la feuille fournie pour répondre à cet exercice, la nature et la valeur des identificateurs utilisés dans le programme suivant aux positions marquées par le gras. Les instructions de la ligne où est marquée la position sont exécutées

avant de déterminer la nature et la valeur des identificateurs quand le programme s'exécute.

```
D.E : Entier nb
D.S : Entier res
Début
Ecrire ("donner nb:")
Lire(nb)
res ← f(nb)  "position1"
Ecrire("f (Entier) =Entier", nb, res)
Fin
Fonction Entier f (Entier n)
Début
Si (n = 1 ) Alors  "position2"
  Retourner (0)
 Sinon  "position3"
Retourner (g (n))
FinSi
FinFct
Entier Fonction g (Entier n) "position4"
Début
Retourner (f (n / 2) + 1)
FinFct
```

La nature de variable est soit une variable globale (notée VG), soit une variable locale (notée VL), soit un paramètre formel (noté PF), soit un paramètre effectif (noté PE).
La valeur sera à déterminer. Par conséquent, une réponse sera un doublet, par exemple (VG, 34), autre exemple (PE, 10).
Répondre à cette question en donnant à nb la valeur 8, et remplir la table ci-dessous par les valeurs correspondantes. Par exemple, la représentation (VG, 0) existe trois fois dans la position 3 de la table, cela veut dire qu'on a passé trois fois par la

position 3 lors de l'exécution et à chaque passage la valeur de **res** est égale à 0.

	nb	**res**	**n**
Position 1	(VG, 8)	.....	
Position 2	.....	.....	.....
Position3	.....	(VG,0) (VG,0) (VG,0)	.....
Position 4	.....	.....	.....

2) Simuler l'exécution de l'algorithme (trouver la valeur de **res** pour nb=8 et pour nb=32).

3) Donner le type de définition des fonctions contenues dans l'algorithme ci-dessus.

4) A quoi sert cet algorithme ? Que fait-il ?

Corrigé de l'examen n°4

Exercice 1 (6 points)

I) Partie1 (Questions de cours) :

a) L'intérêt d'écrire un algorithme c'est de faire le plan d'une solution et de rédiger ensuite le programme correspondant dans un langage donné et parce que l'algorithmique exprime les instructions résolvant un problème donné indépendamment des particularités de tel ou tel langage.

b) On choisit l'instruction "Selon" plutôt que "Si", lorsque les conditions expriment une égalité stricte, permettant de décrire les cas un par un.

c) On utilise "Repéter...jusqu'à" plutôt que "TantQue", lorsque les instructions à exécuter correspondent au cas où il y aurait au moins un traitement à effectuer.

d) Une variable locale est une variable utilisée uniquement au sein de la fonction où elle a été déclarée. Une variable globale c'est une variable déclarée avant le début d'un algorithme et peut être utilisée dans n'importe quelle fonction contenue dans cet algorithme.

e) La boucle "**Pour**" est utilisée lorsque les instructions à exécuter se répètent un nombre de fois fixé à l'avance.

II) Partie 2

Algorithme newton
D.E : Réel V, e

D.S : Réel U
D.I : Réel U0
Début
Ecrire (''donner une valeur positive v'')
Répéter
Lire(V)
Jusqu'à (V>0)
U0=V+1/3*(1/V-V)
Repeter
U=U0+1/3*(V/U0²-U0)
e=(U-U0)/U0
U0=U
Jusqu'à (e<0.00001)
Ecrire ("La racine cubique d'un réel est un réel", V,U)
Fin

Exercice 2 (7 points)

a)
Algorithme Testernombre
D.E : Entier sp,n ,i
Debut
Ecrire('Donner n')
Répéter
Lire(n)
Jusqu'à (n>0)
sp=0
Pour i de 1 à n/2 **Faire**
Si (n%i=0) **Alors**
sp=sp+i
Finpour
Si (n>sp) **Alors**
Ecrire (" Le nombre entier est déficient",n)
Sinon
Si (n=sp) **Alors**

```
    Ecrire (" Le nombre entier est parfait",n)
   Sinon
   Ecrire ( "Le nombre entier est abandant",n)
  Finsi
 Finsi
Finsi
Fin
```

b)

```
Algorithme minchiffre

D.E : Entier n, c
D.S : Entier min
Début
 Répéter
   Ecrire (''donner un nombre n:''), Lire(n)
 Jusqu'à (n>0)
 Faire
 Min←9
  TantQue (n <>0)
  Faire
   c←n%10
   n←n/10
 Si (c<min) Alors
 min←c
Finsi
FinTantQue
Ecrire (''le chiffre minimum du nombre entier  est un entier'', n,
min);
Fin
```

Exercice 3 (7 points)

	nb	res	n
Position 1	(PE, 8) ou (VG, 8)	(VG, 0)	----------
Position 2	(VG, 0)	(VG, 0)	(VL, 1) ou (PF, 1)
Position 3	(VG, 0) (VG, 0) (VG, 0)	(VG,0) (VG,0) (VG,0)	(VL, 8) ou (PF, 8) (VL, 4) ou (PF, 4) (VL, 2) ou (PF, 2)
Position 4	(VG, 0) (VG, 0) (VG, 0)	(VG,0) (VG,0) (VG,0)	(PF, 8) ou (VL, 8) (PF, 4) ou (VL, 4) (PF, 2) ou (VL, 2)

2)

Pour nb =8 → res=3 (0.5 point)

Pour nb=32 →res=5 (0.5 point)

3) le type de définition des fonctions : définition top-down.

4) détermine le nombre de fois où un nombre n est divisible par deux.

Références bibliographiques

- *A.Aho, J. Hopcroft, J. Ullman*. Structures de Données et Algorithmes. InterEditions.
- *C.Froidevaux, M-C. Gaudel, M. Soria*. Types de Données et Algorithmes. Editions McGRAW-Hill.
- Algorithmes et structures de données en langage C. InterEditions 1996.
- Dunod J. Courtin, I. Kowarski. "Initiation à l'Algorithmique et aux Structures de Données",
- Robert CORI et Jean-Jacques LEVY. Algorithmes et Programmation. Ecole Polytechnique.
- Kyle Loudon. Maîtrise des algorithmes en C, O'Reilly (Classique) ; ISBN : 2841770966, 2000
- Michel Divay. Algorithmes et structures de données génériques - Cours et exercices corrigés en langage C. Dunod, 2004.